Titre: Title:	Nouveau protocole de partage de fichiers pair à pair pour réseau ad hoc
Auteur: Author:	Georges Abou-Khalil
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Abou-Khalil, G. (2006). Nouveau protocole de partage de fichiers pair à pair pour réseau ad hoc [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7861/

UNIVERSITÉ DE MONTRÉAL

NOUVEAU PROTOCOLE DE PARTAGE DE FICHIERS PAIR À PAIR POUR RÉSEAU AD HOC

GEORGES ABOU-KHALIL
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION DU DIPLÔME DE
MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AOÛT 2006

Your file Votre référence
ISBN: 978-0-494-19274-0
Our file Notre référence
ISBN: 978-0-494-19274-0

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

NOUVEAU PROTOCOLE DE PARTAGE DE FICHIERS PAIR À PAIR POUR RÉSEAU AD HOC

présenté par : Georges Abou-Khalil
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury constitué de :

M. GALINIER Philippe, Doct., président.
M. PIERRE Samuel, Ph.D., membre et directeur de recherche.
M. QUINTERO Alejandro, Doct., membre.

À mes parents.

Remerciements

J'aimerais tout d'abord remercier mon directeur de recherche, Samuel Pierre, pour son encadrement, ses conseils ainsi que son soutien financier nécessaires à l'accomplissement de mon travail de recherche.

Je voudrais également remercier Gabriel Ioan Ivascu pour m'avoir conseillé et guidé tout au long de mon projet de recherche.

Finalement, j'aimerais remercier tous mes collègues et amis du *LAboratoire de recherche en Réseautique et Informatique Mobile* (LARIM) pour tous les bons moments que nous avons passés ensembles. Je n'oublierai jamais toutes les discussions enrichissantes que nous avons eues.

Résumé

Les réseaux mobiles ad hoc et les réseaux pair à pair partagent plusieurs caractéristiques fondamentales, notamment le manque d'infrastructure, une topologie dynamique ainsi qu'une connexion/déconnexion arbitraire des nœuds. Ces similarités encouragent le développement des réseaux pair à pair pour réseaux ad hoc. Une application pair à pair très utilisée est le partage de fichiers, ayant connu un essor foudroyant sur Internet. Il n'est donc pas surprenant de voir apparaître plusieurs travaux dans la littérature sur le développement d'applications de partage de fichiers pair à pair pour réseaux ad hoc dont le protocole pionner, ORION. Il implémente, pour réseaux ad hoc, les deux aspects requis au fonctionnement d'un protocole de partage de fichiers, soit le mécanisme recherche de fichiers et le mécanisme de transfert de fichiers. Le fonctionnement de ce protocole mimique celui d'un protocole de routage réactif. Ainsi, la recherche de fichiers se fait sur demande, ce qui bâtit progressivement le réseau de recouvrement.

Le travail visé est le développement d'un nouveau protocole de partage de fichiers pair à pair pour réseaux ad hoc basé sur ORION et offrant de meilleures performances au niveau du transfert. L'amélioration découle de l'ajout d'un mécanisme permettant aux nœuds recevant un fichier de desservir les morceaux qu'ils possèdent. Ainsi, avec notre approche, un fichier est desservi par deux types de sources, soient les sources complètes qui le possèdent au complet et les sources partielles qui en possèdent des morceaux. Ce concept a été introduit par Bittorrent, un protocole de partage de fichiers pair à pair pour Internet.

Notre protocole est également plus efficace face aux transferts erronés. En effet, il permet la vérification des données morceau par morceau, contrairement à ORION qui n'effectue la vérification qu'une fois le transfert terminé. De plus, il est plus robuste face aux pannes. Puisque les clients peuvent s'échanger des données, des transferts peuvent continuer à fonctionner même en l'absence de sources complètes pour le fichier. Dans une telle situation avec ORION, les clients se retrouvent bloqués. Finalement, notre protocole permet de gérer les transferts simultanés, offrant la possibilité d'augmenter ou réduire la priorité de certains transferts.

Les résultats obtenus suite à la mise en œuvre de notre solution sont excellents.

Dans le pire cas, la performance de notre protocole est similaire à celle d'ORION. Dans un cas moyen, les performances sont supérieures. Il offre notamment une réduction du temps de transfert des fichiers, une réduction du nombre de paquets dans le réseau ainsi qu'une réduction du débit généré par les nœuds.

Avec de meilleures performances au niveau des transferts, une meilleure gestion des transferts erronés, une meilleure robustesse face aux pannes ainsi qu'un gestionnaire de téléchargement, notre solution s'avère un meilleur choix qu'ORION comme protocole de partage de fichiers pair à pair pour réseaux ad hoc.

Abstract

Mobile ad hoc networks and peer-to-peer networks share many fundamental characteristics, such as the lack of infrastructure, a dynamic topology and an arbitrary connection/disconnection of nodes. These similarities encourage the development of peer-to-peer networks for mobile ad hoc networks. One application of peer-to-peer networks is file sharing. File sharing has gain tremendous popularity on the Internet. A few peer-to-peer protocols for mobile ad hoc networks have been developed, the most noticeable one being ORION. ORION implements the two mechanisms required for file sharing, which are the file searching and the file transferring.

The goal of this work is to implement a new peer-to-peer file sharing protocol for mobile ad hoc networks based on ORION, but offering better performance for the file transferring algorithm. This will be possible with the addition of a mechanism allowing a client receiving a file to serve the pieces he possesses, becoming a partial source for the file. This concept was introduced by Bittorrent, a peer-to-peer protocol for Internet.

Our protocol will also provide better handling of transfer errors. Contrary to ORION, the detection will be done ón a piece by piece basis, allowing for a much smaller retransmission in case of an error. Furthermore, our protocol allows transfers to continue even when no complete sources are available for the file. With ORION, such a scenario would automatically entail the halt of all transfers where as with our protocol; clients can exchange data between them. Finally, our solution offers a simultaneous transfers manager, allowing the user to specify the priority of the transfers.

The results show that our solution offers a much better performance than ORION. It reduces the transfer times, the number of packets in the network and the bandwidth used. In the worst case scenario, its performance is similar to that of ORION.

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et abréviations

7DS	7 Degrees of Seperation
ACK	ACKnowledgement
ANSI	Ad hoc Networking with Swarm Intelligence
AODV	Ad hoc On-demand Distance Vector
BTM	BitTorrent for Manets
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DHT	Distributed Hash Table
DSDV	Destination Sequenced Distance Vector
DSR	Dynamic Source Routing
FIFO	First In First Out
FREF	File REFresh
FREP	File REsPonse
FREQ	File REQuest
FREQK	File REQuest by Keywords
FREQS	File REQuest by Signature
FTP	File Transfer Protocol
GPS	Global Positioning System
GPSR	Greedy Perimeter Stateless Routing
HTTP	HyperText Transfer Protocol
IARP	IntrAzone Routing Protocol
IERP	IntErzone Routing Protocol
IP	Internet Protocol
LRU	Least Recently Used
MAC	Medium Access Control
MACA	Multiple Access with Collision Avoidance
MD5	Message-Digest algorithm 5
MI	Messagerie Instantanée
MPLS	Multiple Path Label Switching
P2P	Peer-to-Peer
PDA	Personal Digital Assistant

PDI	Passive Distributed Indexing
PREQ	Piece REQuest
PREP	Piece REsPonse
PREPC	Piece REsPonse by Complete
PREPI	Piece REsPonse by Incomplete
OLSR	Optimized Link State Routing
ORION	Optimized Routing Independent Overlay Network
OSI	Open Systems Interconnection
RED	Robust Earliest Deadline
RFC	Request For Comment
ROUTEER	ROUTE ERror
RSVP-TE	Resource ReSerVation Protocol for Traffic Engineering
SHA	Secure Hash Algorithm
SMBP	Simple Multicast Broadcast
SREQ	Signatures REQuest
SREP	Signatures REsPonse
TCP	Transmission Control Protocol
TSMA	Time Spread Multiple Access
TTL	Time To Live
UDP	User Datagram Protocol
WFQ	Weighted Fair Queuing
ZRP	Zone Routing Protocol

CHAPITRE 1

INTRODUCTION

Les réseaux pair à pair sont de plus en plus populaires auprès des usagers. Une des applications exploitées est le partage de fichiers qui se fait traditionnellement selon l'architecture Client/Serveur. Les réseaux de partage de fichiers pair à pair et les réseaux mobiles ad hoc aussi partagent plusieurs caractéristiques fondamentales notamment le manque d'infrastructure, une topologie dynamique ainsi qu'une connexion/déconnexion arbitraire des nœuds. Ces similarités encouragent le développement des réseaux de partage pair à pair pour les réseaux mobiles ad hoc, objet principal de ce mémoire.

Dans ce chapitre, nous énoncerons les concepts de base des réseaux de partage pair à pair appliqués aux réseaux mobiles ad hoc, par la suite nous expliciterons les éléments de la problématique, nous énoncerons ensuite les objectifs de ce mémoire ainsi que les principaux résultats attendus. Ce chapitre se terminera par esquisse du plan de ce mémoire.

1.1 Définitions et concepts de base

Un réseau mobile ad hoc (appelé MANET pour Mobile Ad hoc Network) est un réseau auto-configurable de nœuds mobiles, connectés par des liens sans fil, dont l'union forme une topologie arbitraire. Chaque nœud possède une antenne et une portée radio variable. Les nœuds qui sont à portée radio du nœud émetteur sont considérés comme voisins. Les nœuds voisins communiquent directement par onde radio. Un nœud émetteur dont le nœud destination n'est pas un voisin immédiat doit donc collaborer avec ses voisins afin de pouvoir communiquer avec la destination. Ainsi, en plus d'être des clients, les nœuds du réseau jouent également le rôle de routeurs pour leurs pairs. Un tel réseau ne requiert pas d'infrastructure de routage classique pour pouvoir fonctionner.

Les nœuds sont également libres de se déplacer aléatoirement et s'organiser ar-

bitrairement. De plus, l'ajout et le retrait de nœuds se fait de manière aléatoire. Ainsi, la topologie d'un réseau mobile ad hoc peut changer rapidement et de manière imprévisible. Un tel réseau peut fonctionner de manière autonome ou peut être relié à un autre réseau, tel Internet. Les réseaux mobiles ad hoc sont très intéressants lorsqu'il faut déployer un réseau rapidement ou dans des endroits hostiles puisqu'aucune infrastructure n'est requise. La Figure 1.1 illustre un exemple de réseau mobile ad hoc composé d'équipements mobiles comme des téléphones cellulaires, des ordinateurs portables ou des assistants personnels digitaux.

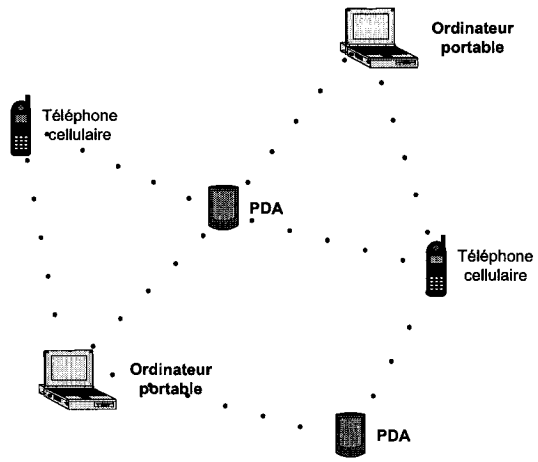
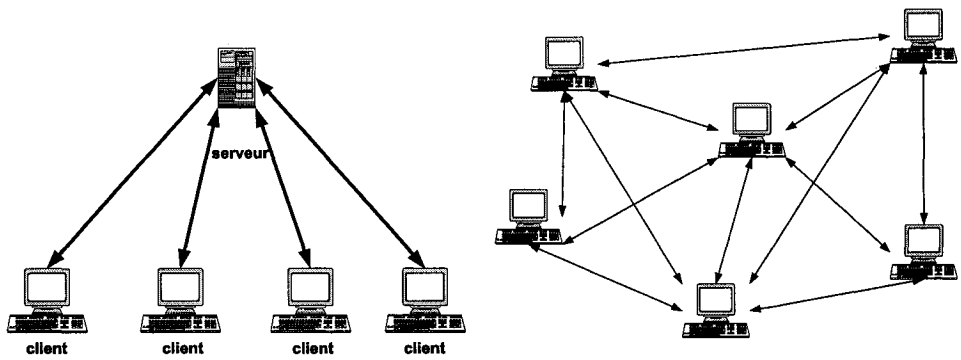


FIGURE 1.1 Exemple de réseau mobile ad hoc

Un réseau informatique pair à pair (P2P) est un réseau qui se base sur la capacité de calcul et sur la bande passante des participants du réseau, contrairement à l'architecture Client/Serveur qui concentre son fonctionnement sur un petit nombre de serveurs. Les Figures 1.2 (a) et (b) illustrent ces deux modèles de réseau de communication. L'avantage des réseaux P2P sur les réseaux Client/Serveur est que l'ajout de nœuds dans un réseau P2P augmente non seulement la demande, mais également la capacité. Les réseaux P2P sont donc plus évolutifs que les réseaux de type Client/Serveur. C'est pour cette raison que les réseaux P2P sont généralement utilisés pour connecter un grand nombre de nœuds de manière ad hoc. Il existe plusieurs applications P2P possibles. L'application traitée dans ce mémoire est le partage de fichiers.



(a) Architecture Client/Serveur (b) Architecture pair à pair

FIGURE 1.2 Deux modèles de communication réseau

1.2 Éléments de la problématique

Un des principaux défis rencontrés dans les réseaux mobiles ad hoc est l'amélioration de l'efficacité du transfert d'information et cela, tout en tenant compte des conditions environnementales difficiles telles que les contraintes énergétiques et la forte mobilité des nœuds. Les avancements dans les communications sans fil doivent réduire les limitations imposées par la diffusion radio. De plus, les protocoles de routage et de transport comme TCP/IP, par exemple, doivent être conçus de manière intelligente de manière à ce que la communication évite de passer par les nœuds qui sont dans un état critique. Un autre défi rencontré est l'amélioration de la rentabilité des réseaux mobiles ad hoc qui devront être en mesure de supporter les applications commerciales futures. En effet, avec l'absence d'une configuration des services réseaux préalables ainsi que d'une autorité centrale, même les tâches de base de ces réseaux deviennent complexes.

Le problème traité dans ce mémoire est le partage de fichiers dans un MANET. La nature ad hoc de ces réseaux facilite l'utilisation des réseaux de partage P2P. L'on considère ici un réseau mobile ad hoc où chaque nœud possède une bibliothèque de fichiers qui peuvent être téléchargés par ses pairs. Lorsqu'un nœud télécharge un fichier, ce fichier est ajouté à sa bibliothèque. Le partage de fichier est composé de deux processus. Le premier est celui qui permet à un usager de localiser un fichier dans le réseau. La recherche se fait généralement à l'aide de mots-clefs qui sont comparés aux noms des fichiers. L'usager peut alors recevoir aucune, une seule ou plusieurs

réponses. Dans le dernier cas, c'est l'usager qui choisit l'exemplaire du fichier qu'il souhaite télécharger. Le fichier peut avoir une ou plusieurs sources. La recherche peut également se faire à l'aide d'une signature. Dans ce cas, la recherche génère au plus un seul résultat, puisque chaque fichier possède une signature unique. Une fois le fichier identifié et la requête de téléchargement envoyée, le deuxième processus est enclenché, soit le transfert du fichier. Cette étape, généralement plus longue, dépend évidemment du nombre de données à transférer, soit la taille du fichier. Plusieurs mécanismes doivent être implémentés afin de s'assurer que le transfert réussisse. Ces mécanismes incluent la détection de pannes, la réparation de routes et la détection de données erronées.

La difficulté du problème vient du fait que même s'il existe plusieurs algorithmes établis pour les réseaux filaires, ils sont peu performants lorsque utilisés dans un MANET. Les caractéristiques des MANET font en sorte qu'il est nécessaire de développer un algorithme de partage de fichiers P2P qui leur est adapté. Un tel algorithme doit donc prendre en considération les défis additionnels que présentent les MANET.

1.3 Objectifs de recherche

L'objectif principal de ce mémoire est de concevoir un système de partage de fichier pair à pair adapté aux caractéristiques des réseaux mobiles ad hoc. Le système proposé implémente les deux processus nécessaires au développement d'un protocole de partage de fichiers P2P, soit le processus de recherche et le processus de transfert. Plus spécifiquement, ce mémoire vise à :

- Analyser les solutions proposées dans la littérature du domaine en matière de réseau de partage pair à pair pour les réseaux mobiles ad hoc ;
- Concevoir et implémenter un système de partage de fichier améliorant les performances des systèmes existant en permettant :
  - aux nœuds qui téléchargent un fichier de desservir les blocs qu'ils possèdent déjà. Ainsi, ce ne sont pas uniquement les nœuds qui possèdent le ficher en sa totalité qui le desservent, mais également ceux qui en possèdent des parties ;
  - aux nœuds de changer dynamiquement leur source lorsqu'ils téléchargent un fichier, réduisant ainsi le nombre de transmissions et augmentant l'efficacité du réseau lorsque de nouveaux transferts sont déclenchés ;
  - une meilleure sécurité afin de pouvoir détecter les erreurs de transmissiosn

plus rapidement et de les corriger avec une plus faible retransmission ;
- une meilleure robustesse face à la disparition de sources en permettant aux clients de continuer à s'échanger des morceaux même si aucune source complète n'est présente dans le réseau ;
- aux nœuds de gérer des transferts simultanés et d'établir des priorités entre ces derniers. Ainsi, un transfert prioritaire va recevoir davantage de bande passante qu'un transfert de moindre priorité.

- Évaluer et analyser la performance des algorithmes et mécanismes proposés en les comparants aux meilleurs systèmes recensés dans la littérature.

Ces améliorations nous permettront d'obtenir un protocole plus performant qui va diminuera le nombre de transmissions globales dans le réseau et qui réduira et les délais de transfert, et le niveau d'énergie dépensée par les nœuds.

1.4 Plan du mémoire

Ce mémoire est composé de cinq chapitres. Après ce chapitre d'introduction, le deuxième chapitre propose un état de l'art sélectif, exposant les concepts utilisés tout au long de ce mémoire ainsi que les principales solutions existantes. Le troisième chapitre donne les détails de la solution proposée. Le quatrième chapitre présente les simulations réalisées ainsi que les résultats obtenus. Le cinquième chapitre conclut ce mémoire en faisant une synthèse du travail effectué ainsi qu'en présentant les idées de travaux futurs pouvant faire l'objet de travaux de recherche.

CHAPITRE 2

CARACTÉRISATION DES RÉSEAUX MOBILES AD HOC ET DES RÉSEAUX PAIR À PAIR

Un réseau mobile sans-fil ad hoc (appelé MANET pour Mobile Ad hoc Network) est un réseau auto-configurable de nœuds mobiles connectés par des liens sans fil, dont l'union forme une topologie arbitraire. Un réseau informatique pair à pair (P2P) est un réseau qui se base sur la capacité de calcul et sur la bande passante des participants du réseau, contrairement à l'architecture Client/Serveur qui concentre son fonctionnement sur un petit nombre de serveurs. Les caractéristiques des MANET font en sorte qu'il est nécessaire de développer des algorithmes de partage de fichiers P2P qui leur sont adaptés. De tels algorithmes doivent donc prendre en considération les défis additionnels que présentent les MANETs. Le présent chapitre propose, dans un premier temps, de définir les concepts propres aux réseaux mobiles ad hoc en plus d'en présenter les caractéristiques et applications. Dans un deuxième temps, nous présentons un état de l'art sur les différentes technologies utilisées dans les réseaux de partage pair à pair.

2.1 Types d'équipements

Un réseau mobile ad hoc est constitué de nœuds équipés d'émetteurs et de récepteurs sans-fil fonctionnant avec une antenne qui peut être omnidirectionnelle (broadcast), très directionnelle (point à point), possiblement dirigeable ou une combinaison de ces caractéristiques. Les types de nœuds les plus courants incluent les téléphones portables, les ordinateurs portables ainsi les assistants personnels digitaux.

2.2 Caractéristiques et utilisation des réseaux mobiles ad hoc

Les réseaux mobiles ad hoc présentent des caractéristiques particulières qui les distinguent des réseaux traditionnels basés sur une infrastructure de routage. Ces caractéristiques sont (Perkins et Hughes, 2002; S. Corson, 1999) :

- Une topologie dynamique qui varie aléatoirement dans le temps. Ceci découle du fait que les nœuds sont libres de se déplacer, les liens étant constamment modifiés ;
- Une bande passante limitée et des liens avec capacités variables. Les liens sans-fil ont une capacité inférieure aux liens câblés. De plus, le débit des communications sans fil, en considérant les effets des accès multiples, de l'effacement (*fading*), du bruit, des interférences et autres, est souvent inférieur aux capacités théoriques des liaisons radios ;
- Des contraintes d'énergies. La plupart des nœuds d'un réseau mobile ad hoc vont opérer sur des batteries ou autre ressource épuisable d'énergie. Pour ces nœuds, il est important que la conception du système soit effectuée avec comme priorité la conservation énergétique ;
- Une sécurité physique limitée. Le réseaux mobiles sans fil sont généralement plus susceptibles aux attaques physiques de sécurité que les réseaux filaires. La possibilité plus élevée de l'écoute électronique (*eavesdropping*), mystification (*spoofing*) et déni de service doit être considérée.

Certaines applications des réseaux ad hoc incluent des applications industrielles et commerciales impliquant des échanges mobiles de données. Il existe également des besoins dans le domaine militaire pour des services de données robustes à l'intérieur des réseaux de communication mobiles sans fil. De plus, le développement de technologies « portables »peut fournir des applications pour les réseaux ad hoc. Lorsque combinés efficacement avec un système de livraison d'information basé sur satellite, les réseaux ad hoc peuvent fournir une méthode extrêmement flexible pour établir une communication dans les opérations de rescousse ou autres scénarios qui requièrent un déploiement rapide des équipements de communication.

2.3 Le routage : principes et problèmes

Le routage est le processus qui permet d'acheminer l'information à travers un réseau, partant d'un nœud source vers un nœud destination. Le chemin que l'information emprunte contient typiquement un ou plusieurs nœuds intermédiaires. Le routage est souvent confondu avec le *bridging*, deux processus qui semblent accomplir la même tâche aux yeux d'un observateur non informé. La différence primaire entre les deux est que le *bridging* est effectué au niveau de la couche 2 (la couche liaison) du modèle OSI alors que le routage est effectué au niveau de la couche 3 (la couche réseau). Cette distinction fournit à ces deux processus des informations différentes à utiliser pour acheminer les paquets entre d'une source à une destination. Ainsi, ces deux processus accomplissent leurs tâches de manières différentes.

Le routage est un sujet qui a été couvert dans la littérature scientifique pendant plus de deux décennies. Cependant, sa popularité commerciale ne date que de la fin des années 80. La raison primaire est que les réseaux des années 70 étaient des environnements simples et homogènes. L'interconnexion à haute échelle est un phénomène assez récent (surtout avec l'apparition d'Internet).

Le routage implique deux processus de base, soit la détermination de routes optimales suivi de l'acheminement de groupes d'information (appelées paquets) à travers le réseau jusqu'à une ou plusieurs destinations. Ce dernier processus est souvent appelé commutation de paquets. Même si la commutation de paquets est un processus assez simple, la découverte de routes ne l'est pas, ce qui rend le routage complexe.

2.4 Découverte de routes

Les protocoles de routage utilisent des métriques pour évaluer le meilleur chemin qu'un paquet peut emprunter. Une métrique est une mesure, la bande passante d'un chemin, par exemple, utilisée par les algorithmes de routages pour déterminer les chemins optimaux. Pour aider le processus de découverte de chemin, les algorithmes de routage initialisent et maintiennent des tables de routage qui contiennent de l'information sur les routes. Ces informations varient en fonction de l'algorithme utilisé. Par exemple, des associations « Destination/Prochain Saut » informent un routeur qu'une destination peut être rejointe de manière optimale en acheminant le paquet vers un routeur particulier représentant le prochain saut sur le chemin vers la des-

tination. Lorsqu'un routeur reçoit un paquet, il vérifie l'adresse de la destination et tente d'associer cette adresse à un prochain saut. La Figure 2.1 illustre un exemple de ce phénomène.

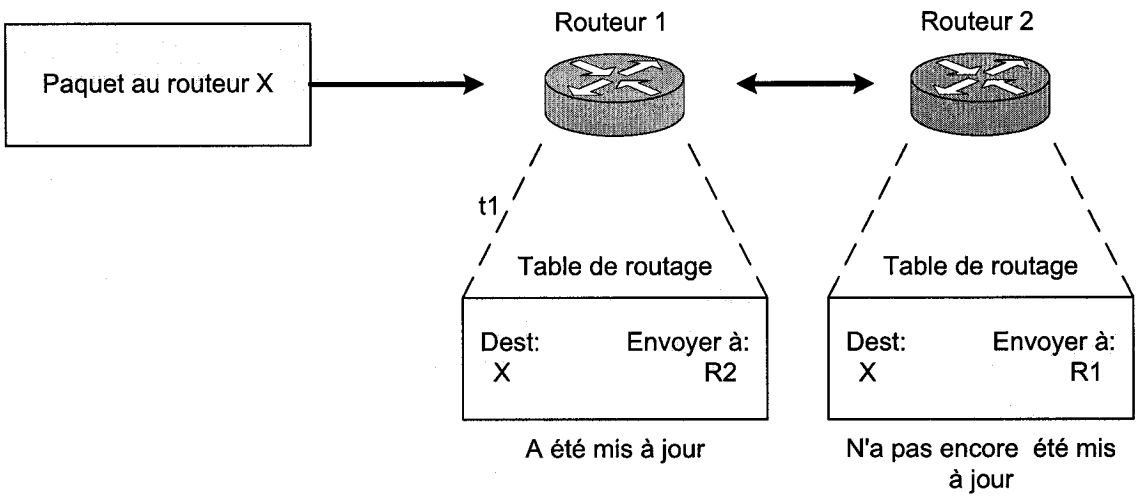


FIGURE 2.1 Exemple de prochain saut

Les routeurs vont utiliser les métriques pour déterminer les routes optimales et les métriques vont différer selon la conception de l'algorithme de routage. Les routeurs communiquent entre eux et maintiennent leurs tables de routage en s'échangeant différents types de messages. Par exemple, les messages de mise à jour consistent généralement en une portion ou en la totalité d'une table de routage. En analysant les mises à jour émises par les autres routeurs, un routeur peut bâtir une image détaillée de la topologie du réseau. Un autre exemple de message est le message état des liens qui va informer les routeurs de l'état des liens de l'émetteur. Ceux-ci sont également importants lorsqu'il s'agit de bâtir la topologie du réseau et lors de la recherche des chemins optimaux.

2.5 Commutation de paquets

Les algorithmes de commutation sont relativement simples et sont identiques pour la plupart des protocoles de routages. Le processus débute lorsqu'un nœud détermine qu'il doit envoyer un paquet vers un autre nœud. Ayant acquis l'adresse d'un rou-

teur, le nœud source envoie un paquet destiné spécifiquement à l'adresse physique du routeur avec comme adresse de couche réseau l'adresse du nœud destination. Ensuite, le routeur examine cette adresse et vérifie s'il connaît le prochain saut vers la destination. Le cas échéant, le routeur change l'adresse physique de la destination vers le prochain saut et transmet le paquet. Sinon, il rejette généralement le paquet. Le prochain saut peut être la destination finale. Dans le cas contraire, le prochain saut est généralement un autre routeur qui recommence le même processus. Ainsi, tout au long du parcourt, l'adresse physique du paquet est modifiée à chaque saut, mais son adresse réseau reste identique.

2.5.1 Algorithmes de routage

Les algorithmes de routage peuvent être différenciés selon certaines caractéristiques. Ce sont les objectifs particuliers de l'algorithme qui guident la conception. Chaque algorithme a un impact différent sur le réseau et les ressources du routeur. Les algorithmes de routage ont souvent un ou plusieurs des objectifs suivants [2] :

- Optimalité : L'optimalité réfère à la capacité de l'algorithme de sélectionner la meilleure route qui dépend des métriques utilisés lors des calculs. Par exemple, un algorithme peut utiliser le nombre de sauts ainsi que le délai comme métriques, mais en attribuant un plus gros poids au délai lors des calculs ;
- Simplicité et faible surdébit : Les algorithmes de routage sont également conçus de la manière la plus simple possible. Ainsi, l'algorithme doit offrir ses fonctionnalités de manière efficace, avec un minimum de logiciels et de surdébit ;
- Robustesse et stabilité : Les algorithmes de routage doivent être robustes, c'est-à-dire qu'ils doivent performer correctement dans des situations inhabituelles et imprévues, comme des pannes matérielles, des congestions ou encore des implémentations incorrectes. Comme les routeurs sont placés aux points de jonctions des réseaux, les conséquences sont graves s'ils ne fonctionnent pas correctement ;
- Convergence rapide : Les algorithmes de routage doivent converger rapidement. La convergence est le processus par lequel tous les routeurs se mettent d'accord sur les routes optimales vers une destination. Ainsi, lorsqu'une panne survient, les routeurs distribuent les messages de mise à jour qui se propagent dans les réseaux, stimulant ainsi un nouveau calcul des routes optimales par les routeurs.

Les algorithmes de routage qui convergent lentement vont entraîner des boucles de routages ou des pannes de réseau ;

- Flexibilité : Les algorithmes de routage doivent également être flexibles, ce qui signifie qu'ils doivent rapidement s'adapter aux différents évènements imprévus du réseau. Par exemple, si un segment du réseau tombe en panne, les algorithmes doivent sélectionner le prochain meilleur chemin pour toutes les routes qui utilisent ce segment. Les algorithmes de routages peuvent être programmés pour s'adapter aux changements de bande passante du réseau, aux tailles des files d'attentes des routeurs, aux délais, par exemple.

2.5.2 Types d'algorithmes de routage

Les algorithmes de routage possèdent plusieurs caractéristiques qui les distinguent. Un algorithme peut être statique ou dynamique. Les algorithmes statiques ne sont pas véritablement des algorithmes mais plutôt des tables de correspondances établies par l'administrateur du réseau avant le début du routage. Ces correspondances ne peuvent pas être modifiées dynamique, l'administrateur doit les modifier explicitement. Ces tables sont simples à concevoir et fonctionnent bien dans des réseaux où le trafic est prévisible et dont la conception est relativement simple. Comme les systèmes de routage statique ne peuvent réagir aux changements dans les réseaux, ils sont généralement considérés inadéquats pour les réseaux d'aujourd'hui, qui sont larges et modifiés continuellement.

La plupart des algorithmes de routages utilisés de nos jours sont dynamiques et s'ajustent aux changements du réseau en analysant les messages de mise à jour de routage. Si le message indique qu'un changement dans le réseau s'est produit, l'algorithme de routage recalcule les routes et envoie de nouveaux messages de mise à jour. Ces messages sont diffusés à travers le réseau, stimulant ainsi les autres routeurs à recalculer leurs routes et modifier leurs tables de routage. Le routage dynamique peut être supplémenté de routes statiques, si nécessaire. On peut, par exemple, désigner un routeur de dernier recourt vers lequel tous les paquets qui ne peuvent être acheminés seront envoyés et qui va agir comme dépôt pour ces paquets. Ceci assure que tous les messages sont traités.

Certains algorithmes de routage supportent plusieurs chemins vers une même destination. Contrairement aux algorithmes de routage avec chemins simples, ils per-

mettent le multiplexage du trafic sur plusieurs lignes. Les avantages sont évidents : ils permettent un meilleur débit et une meilleure fiabilité. Le multiplexage est généralement appelé répartition de charge. Les algorithmes multi-chemins sont naturellement plus complexes à développer.

Certains algorithmes de routage opèrent dans un espace plat, alors que d'autres font du routage hiérarchique. Dans un système de routage plat, tous les routeurs sont des pairs égaux entre eux. Dans un système hiérarchique, certains routeurs forment un réseau dorsal de routage. Les paquets qui ne proviennent pas du réseau dorsal sont acheminés aux routeurs du réseau dorsal et le traversent jusqu'à ce que la zone générale dans laquelle se trouve la destination soit atteinte. À ce point, ils sont acheminés du dernier routeur du réseau dorsal à la destination finale, en passant par un ou plusieurs routeurs qui ne font pas partie du réseau dorsal. Les systèmes de routage vont souvent designer des groupes logiques de nœuds, appelés domaines, systèmes autonomes ou régions. Dans les systèmes hiérarchiques, certains routeurs dans un domaine peuvent communiquer avec des routeurs d'autres domaines, alors que d'autres ne peuvent communiquer qu'avec des routeurs du même domaine. Dans des réseaux très larges, des niveaux hiérarchiques additionnels peuvent exister, avec les routeurs situés au niveau le plus haut formant le routage dorsal. L'avantage premier d'un routage hiérarchique est qu'il mimique l'organisation des compagnies et donc supporte également leur modèle de trafic. La plupart des communications dans le réseau s'effectuent entre les petits groupes de compagnies (domaines). Puisque les routeurs intra-domaines ont uniquement besoin de connaître les autres routeurs dans leur domaine, leur algorithme de routage peut être simplifié et, dépendamment de l'algorithme de routage utilisé, le trafic de mise à jour du routage peut être réduit en conséquence. Des exemples d'algorithmes hiérarchiques incluent FSR (Pei *et al.*, 2000) et GSRP (Chen et Gerla, 1998).

Certains algorithmes de routage supposent que le nœud source va déterminer la route au complet. Ce phénomène est appelé routage à la source (*source routing*). Dans de tels systèmes, les routeurs vont agir comme de simples relais pour les paquets, envoyant chaque paquet reçu à la prochaine destination. D'autres algorithmes supposent que les nœuds sources ne connaissent rien à propos des routes. Dans ces algorithmes, ce sont les routeurs qui déterminent le chemin à emprunter, d'après leurs calculs. Dans le premier système, l'intelligence de routage se trouve au niveau des nœuds sources alors que dans le deuxième, elle se trouve dans les routeurs. Un exemple

d'algorithme de routage à la source est DSR (Johnson et Maltz, 1996). Certains algorithmes de routage fonctionnent uniquement à l'intérieur des domaines, alors que d'autres fonctionnent à l'intérieur et entre les domaines. La nature de ces deux types d'algorithmes est différente. Il en découle qu'un algorithme intra-domaine optimal ne reste pas nécessairement optimal lorsque utilisé comme algorithme inter-domaine. Un bon exemple est le protocole ZRP (Haas, 1997), qui est en fait une combinaison d'un protocole inter-domaine et un protocole intra-domaine.

Les algorithmes de type états des liens (également connus sous le nom d'algorithmes du plus court chemin) inondent le réseau d'informations de routage. Chaque routeur envoie uniquement une portion de sa table de routage qui décrit l'état de ses liens. Ainsi, chaque routeur bâtit une image du réseau entier dans ses tables de routage. Les algorithmes de type vecteur distance (également connus sous le nom d'algorithmes *Bellman-Ford*) voient à ce que chaque routeur envoie tout ou une partie de sa table de routage à ses voisins. Les algorithmes de type état des liens utilisent donc de petites mises à jour diffusées à travers tout le réseau alors que les algorithmes de type vecteur distance utilisent des mises à jour plus complètes mais envoyées uniquement aux routeurs voisins. Les algorithmes de type vecteur distance ne connaissent donc que leurs voisins. Puisqu'ils convergent plus rapidement, les algorithmes de type état des liens sont moins susceptibles aux boucles que les algorithmes de type vecteur distance. Cependant, ils requièrent davantage de ressources du processeur et de la mémoire et sont donc plus dispendieux à implanter et supporter. Les algorithmes de type état des liens sont également plus évolutifs que les protocoles de type vecteur distance. Un exemple d'un protocole de type vecteur distance est DSDV (Perkins et Bhagwat, 1994) alors qu'un exemple d'un protocole de type états des liens est OLSR (Jacquet *et al.*, 2001).

2.6 Routage dans les réseaux ad hoc

Les caractéristiques uniques aux réseaux mobiles ad hoc compliquent davantage le processus de routage dans ces réseaux. L'absence d'infrastructure fait en sorte que les nœuds doivent, en plus d'envoyer et de recevoir des données, acheminer les données provenant des autres nœuds du réseau. Ils jouent également le rôle des routeurs. Cela ajoute une plus grosse charge de travail sur les nœuds, ce qui entraîne une consommation plus grande d'énergie. Or, une des contraintes les plus dures des réseaux ad

hoc est que leur source d'énergie est généralement limitée. Il faut donc trouver un compromis. De plus, le fait que les nœuds servent de routeurs soulève également le problème de collaboration des nœuds. En effet, si un nœud décide de ne pas acheminer les paquets des autres nœuds (car il n'y voit pas d'intérêt personnel), la performance du réseau est réduite.

2.7 Taxonomie des protocoles de routage

Initialement, les protocoles de routage pour réseaux mobiles ad hoc étaient classés en deux grandes catégories, soit les protocoles proactifs et les protocoles réactifs. Depuis, d'autres types de protocoles ont vu le jour. Les sections suivantes passent en revue les différents types de protocoles et exposent les protocoles les plus étudiés dans la littérature.

2.7.1 Protocoles proactifs

Les protocoles proactifs (*table-driven*) représentent les premiers types de protocoles de routage pour MANET. Ces protocoles maintiennent une vue d'ensemble cohérente du réseau. Chaque nœud possède une table de routage dans laquelle il maintient l'information sur les emplacements des autres nœuds du réseau. Cette information est utilisée lors de la recherche de routes. Pour s'assurer que la table de routage reste à jour, ces protocoles adoptent différents mécanismes. Un de ces mécanismes est la diffusion d'un message « hello », message spécial contenant l'adresse du nœud, transmis à des intervalles réguliers. Lors de la réception de ce message, chaque nœud rafraîchit sa table de routage avec les nouvelles informations reçues.

Les protocoles de routage proactifs peuvent être inadéquats pour les MANET. En effet, les nœuds opèrent souvent avec une puissance et une bande passante limitée. Ainsi, si le réseau affiche une forte mobilité, les rafraîchissements fréquents des tables de routage vont dépenser énormément de puissance et de bande passante, ce qui peut entraîner une congestion dans le réseau. Les protocoles de routage proactif sont donc mieux adaptés aux réseaux peu mobiles. Un exemple d'algorithme proactif est DSDV. Cet algorithme est de type vecteur distance. Chaque nœud maintient une liste de toutes les destinations et le nombre de sauts requis pour atteindre chacune de ces destinations. Le principal désavantage de cette solution est qu'il faut régler un bon

nombre de paramètres, tel que l'intervalle du rafraîchissement périodique et le nombre d'intervalles de rafraîchissement qui peuvent s'écouler avant de considérer une route comme non valide.

Un autre algorithme proactif intéressant est OLSR. Dans cet algorithme, certains nœuds dont désignés comme relais multipoints par leurs voisins. Ils vont alors diffuser les informations relatives à la topologie régulièrement dans leurs messages de contrôle.

2.7.2 Protocoles réactifs

Les protocoles de routage réactifs (*on-demand*) sont apparus après les protocoles proactifs. Dans les protocoles de routage réactifs, on ne garde aucune information sur les nœuds. Ainsi, lorsqu'un nœud recherche une route vers une destination dont il ne connaît pas le chemin, il initie un processus de recherche de route. Un paquet de découverte de route est alors envoyé vers ses voisins et progresse de nœud en nœud jusqu'à atteindre la destination ou un nœud intermédiaire qui possède une route vers la destination. Une fois que la destination reçoit tous les chemins possibles, elle renvoie à la source la meilleure route. Le nœud source va alors utiliser ce chemin pour communiquer avec la destination. De plus, il garde la route en mémoire pour une utilisation ultérieure éventuelle.

Un exemple d'algorithme réactif est DSR. Comme tout algorithme réactif, une route est calculée sur demande, lorsqu'un nœud transmetteur le demande. Lorsqu'une route est calculée, celle-ci va contenir la liste de tous les nœuds intermédiaires entre la source et la destination. Le principal désavantage de cette méthode survient lorsque les routes sont très longues ou bien lorsque les adresses sont grandes (IPv6 par exemple). On obtient alors un énorme sudébit dans les paquets. Pour remédier à ce problème, DSR permet de définir un flow id permettant d'acheminer les paquets saut par saut.

Un autre algorithme réactif est AODV (Perkins et Royer, 1999). AODV permet de réduire la signalisation dans le réseau par rapport à DSR. Cela découle du fait que chaque nœud possède une table de routage. Lorsqu'une route est créée, chaque nœud intermédiaire enregistre la route dans sa table de routage. Ainsi, lorsqu'un nœud désire transmettre, il va d'abord vérifier s'il ne possède pas de route fraîche dans sa table. Le cas échéant, aucune découverte de route n'est nécessaire. Évidemment, pour limiter la taille de cette table, une mise à jour régulière est effectuée afin de purger les routes non utilisées. AODV est souvent utilisé comme algorithme de référence

pour les nouveaux protocoles développés, car il est considéré comme efficace dans les réseaux ad hoc typiques.

2.7.3 Protocoles hybrides

Les protocoles de routage hybrides tentent de combiner les meilleures caractéristiques des protocoles proactifs et les protocoles réactifs afin de profiter des avantages de chacun. Ils vont donc tenter de régler le problème de latence des protocoles réactifs ainsi que les problèmes de surdébit des protocoles proactifs.

Le protocole hybride le plus connu est ZRP. Ce protocole fonctionne en divisant le réseau en zones de routage distinctes. Deux protocoles indépendants sont alors utilisés, un protocole qui opère à l'intérieur des zones et un protocole qui opère entre les zones. Le protocole intra-zone (IARP) est un protocole proactif. Ainsi, chaque nœud connaît la topologie de la zone qui le contient. Le protocole inter-zone (IERP) est un protocole réactif. Ainsi, si un nœud doit communiquer avec un autre nœud qui ne se trouve pas dans sa zone, une requête de découverte de route est lancée. Le diamètre des zones est une variable qui doit être choisie dépendamment de la topologie du réseau. Des techniques existent pour calculer de bonnes estimations du diamètre, jusqu'à 2% de la valeur optimale.

2.7.4 Protocoles géographiques

Les algorithmes de routage géographiques sont une famille de protocoles qui utilisent l'emplacement géographique des nœuds pour effectuer le routage. Contrairement aux algorithmes proactifs ou réactifs, ils ne font pas appel à la topologie du réseau. Ce sont les emplacements des nœuds qui sont utilisés comme adresses. L'acheminement des paquets se fait de manière gloutonne. Ces protocoles supposent que chaque nœud puisse connaître son emplacement physique (à l'aide d'un GPS, par exemple). Les algorithmes de routage géographiques promettent une meilleure extensibilité par rapport aux algorithmes basés sur la topologie.

Le protocole géographique le plus connu est GPSR (Karp et Kung, 2000). Il utilise la position géographique des nœuds pour acheminer les paquets. À chaque saut, le protocole s'assure d'envoyer le paquet au nœud le plus proche de la destination. Il s'agit donc d'un algorithme glouton qui transmet les paquets aux nœuds qui sont progressivement plus proches de la destination. Dans le cas où aucun voisin du nœud

n'est plus proche que lui de la destination, GPSR doit effectuer un acheminement en mode périmètre. Dans ce cas, le paquet traverse successivement les nœuds à droite du nœud (algorithme de la main droite) jusqu'à trouver un nœud plus proche de la destination. C'est alors que l'algorithme glouton peut continuer son travail.

2.8 Les réseaux pair à pair

Un réseau informatique pair à pair (P2P) est un réseau qui compte sur la capacité de calcul et sur la bande passante des participants du réseau, contrairement à l'architecture Client/Serveur qui concentre son fonctionnement sur un petit nombre de serveurs. Les réseaux P2P sont généralement utilisés pour connecter un grand nombre de nœuds de manière ad hoc. Les caractéristiques des réseaux P2P sont les suivantes :

- Transmission en temps réel de données et de messages à l'aide de connexions directes entre les pairs ;
- Les pairs jouent à la fois le rôle de clients et de serveurs ;
- Ce sont les pairs qui fournissent le contenu ;
- Les pairs sont autonomes mais exercent un certain contrôle sur les autres pairs ;
- Des connexions variables et des adresses temporaires.

2.9 Comparaison avec le modèle Client/Serveur

Les réseaux P2P offrent plusieurs avantages par rapport au modèle Client/Serveur, ce qui les rend très intéressants. Ces avantages sont :

- *Une plus grande évolutivité.* Tous les clients fournissent des ressources telle la bande passante, l'espace de stockage et la puissance de calcul. Ainsi, même si l'arrivée d'un nœud augmente la demande, elle augmente également la capacité totale du réseau. Cela n'est certainement pas le cas pour le modèle Client/Serveur qui, avec un nombre fixe de serveurs, voit ses capacités diminuer avec l'augmentation du nombre de clients (ce qui détériore généralement le service) ;
- *Une meilleure résistance aux pannes.* Grâce à leur nature distribuée, les réseaux P2P offre une meilleure robustesse en cas de panne puisque les données sont répliquées à travers plusieurs nœuds et, dans le cas de réseaux P2P purs, en

permettant aux pairs de trouver l'information sans passer par un serveur central ;

- *Une meilleure confidentialité.* Comme la communication entre les nœuds ne passent pas par un serveur centralisé mais plutôt par un lien direct, il y a moins de chances que la communication soit interceptée ;
- *Flexibilité et adaptabilité.* Les réseaux P2P requièrent peu ou pas d'infrastructures et sont donc plus facilement déployables et modifiables.

Cependant, il existe également des inconvénients à l'utilisation de réseaux P2P. Ces inconvénients sont :

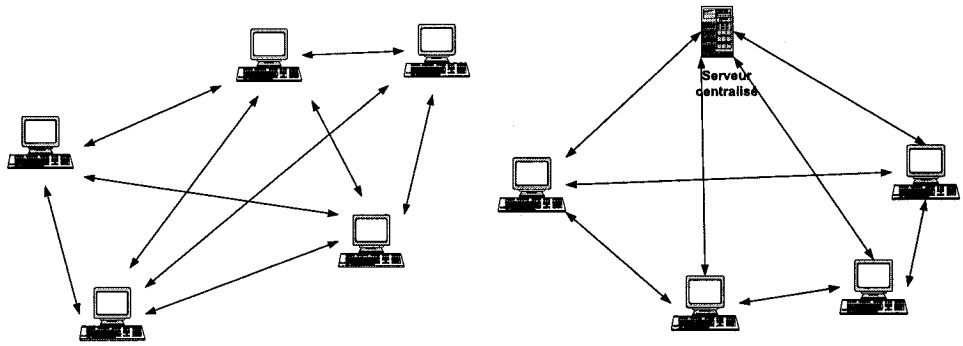
- *Aucune garantie sur la disponibilité d'une ressource.* Puisque les ressources sont dispersées à travers les pairs, rien ne garantit que les pairs qui possèdent la ressource désirée soient disponibles pour la desservir ;
- *Explosion du trafic.* Comme la plupart des messages sont diffusés à travers le réseau, la bande passante risque d'être saturée avec la signalisation et il y a donc diminution de la bande passante dédiée aux données utiles ;
- *Une plus grande exposition aux attaques et aux virus.* Comme il n'y a pas d'entités centrales contrôlant les transmissions et puisqu'il y a accès direct, il est plus facile pour un pair d'attaquer les autres pairs du réseau.

2.10 Types de réseaux P2P

Les réseaux P2P existent sous deux grandes formes, soit les réseaux P2P purs et les réseaux P2P hybrides. La Figure 2.2 illustre ces deux types de réseaux P2P. Un réseau P2P pur n'utilise pas la notion de clients ou serveurs. Tous les nœuds sont égaux entre eux, sont directement connectés entre eux et jouent simultanément les rôles de serveurs et clients vis-à-vis des autres nœuds du réseau. Il s'agit donc d'une architecture plate qui diffère significativement du modèle Client/Serveur. Le processus de découverte est généralement réalisé à l'aide d'une diffusion générale (*broadcast*) ou d'une multidiffusion sélective (*multicast*). Dans des réseaux où aucun type de diffusion n'est possible comme l'Internet, par exemple, d'autres méthodes doivent être utilisées.

Un réseau P2P hybride contient un serveur central qui garde certaines informations relatives aux nœuds et qui dessert ces informations aux nœuds intéressés. Il existe trois types de serveurs :

- *Un serveur simple de découverte.* Dans cette configuration, le serveur central



(a) Réseau P2P pur (b) Réseau P2P hybride

FIGURE 2.2 Types de réseau P2P

est utilisé pour découvrir les autres pairs du réseau. Généralement, l'implémentation est faite de sorte que lorsque l'application P2P démarre sur un pair, elle s'enregistre auprès du serveur. Lorsqu'un pair veut entrer en contact avec un autre, il doit demander au serveur la liste de tous les pairs et contacter directement chacun des pairs de la liste pour trouver celui qu'il recherche ;

- *Un serveur de découverte et de requête.* Dans cette configuration, le serveur joue le même rôle que dans le cas précédent mais il administre en plus la liste des contenus de chaque pair lorsqu'ils s'enregistrent auprès du serveur. Ainsi, lorsqu'un pair recherche une ressource, il va transmettre la requête au serveur qui va lui retourner l'adresse du ou des nœuds qui possèdent la ressource ;
- *Un serveur de découverte, de requête et de contenu.* Dans cette configuration, le serveur contient également les ressources à partager. Ces ressources lui sont transmises par les nœuds lorsque l'application démarre. Cette approche montre comment un réseau P2P peut s'approcher du modèle de réseau Client/Serveur.

On peut également envisager des réseaux P2P mixtes qui fonctionnent comme des réseaux P2P hybrides mais, dans le cas où le serveur n'est plus disponible, leur fonctionnement évolue vers celui d'un réseau P2P pur.

2.11 Fonctionnement des réseaux P2P

Tous les réseaux P2P ont des fonctionnalités de base commune. Dans cette section, nous présenterons différents mécanismes utilisés par les réseaux P2P : la technique

des réseaux de recouvrement, les stratégies de découverte et finalement, l'adressage des nœuds.

Les réseaux de recouvrement Un protocole P2P fonctionne en créant un réseau de recouvrement (*overlay network*), opérant au dessus le réseau original. Dans un réseau de recouvrement, les nœuds sont reliés par des liens virtuels ou logiques, chacun de ces liens correspondant à un chemin qui peut passer par plusieurs liens physiques du réseau original. Les réseaux de recouvrement créent donc une topologie virtuelle au dessus la topologie physique. Les nombreux réseaux de recouvrement P2P qui ont été proposés peuvent être groupés en deux catégories, soit les réseaux de recouvrement P2P non structurés et les réseaux de recouvrement P2P structurés.

Les réseaux de recouvrement P2P non structurés n'ont pas de contrôle précis sur la topologie de recouvrement. Le réseau est typiquement formé de nœuds qui se joignent au réseau d'une façon prédéterminée. La topologie du réseau résultat suit certaines formes comme une loi de puissance, par exemple, mais l'emplacement des ressources n'est pas basé sur la topologie du réseau. De plus, ce type de réseau ne tient généralement pas compte de la proximité des nœuds. Ainsi, des nœuds voisins dans le réseau de recouvrement peuvent être très distants dans le réseau de base. La méthode typique de recherche dans un tel réseau se fait par la diffusion de la requête à travers le réseau, en la limitant à l'intérieur d'un certain rayon. Même s'il existe des méthodes plus efficaces qui ont été développées pour des recherches plus performantes, la diffusion des requêtes reste le point faible des réseaux non structurés. Malgré ce point faible, ce type de réseau présente un avantage très intéressant : sa robustesse. Effectivement, l'ajout et le départ de nœuds n'apporte que des modifications mineures à la topologie du réseau, qui continue à fonctionner sans nécessiter une reconfiguration majeure.

Pour combler les faiblesses de la recherche de ressources dans les réseaux non structurés, les réseaux de recouvrement structurés ont été proposés. Ces derniers combinent l'auto-configuration, la décentralisation et la diversité des réseaux non structurés à un algorithme de routage efficace et évolutif qui est capable de localiser de manière sûre des ressources, sans dépasser un nombre maximal de sauts (typiquement algorithmique par rapport à la taille du réseau), et cela tout en exploitant les propriétés de proximité du réseau original. De nombreux types réseaux de recouvrement P2P structurés ont été proposés. Le routage de ces réseaux implémente de

manière efficace des tables de hachage distribuées (DHT) qui sont évolutives et insensibles aux défaillances : chaque nœud du réseau possède un identificateur unique (nodeID) et chaque ressource dans le réseau possède une clef unique. Les nodeID et les clefs existent dans le même espace de nommage (namespace) et un message avec une clef est associé à un nœud unique dans le réseau de recouvrement. Ainsi, les DHT permettent l'ajout de ressources sans avoir besoin de connaître l'endroit où elles seront stockées. Ils permettent également le routage de requêtes sans nécessiter de connaître l'emplacement des données recherchées. Cependant, pour pouvoir bien fonctionner, chaque nœud d'un réseau de recouvrement structuré doit maintenir une liste de voisins qui satisfont certains critères. Pour cette raison, ce type de réseau est moins résilient à l'arrivée et aux départs fréquents de nœuds.

Stratégies de découverte La première étape qu'un nœud doit réaliser pour participer dans un réseau P2P est de se joindre au réseau de recouvrement. Pour ce faire, le nœud doit pouvoir découvrir au moins un nœud de ce réseau. Il existe plusieurs stratégies de découverte de nœuds :

- *Voisins statiques.* Avec cette stratégie, chaque nœud possède une liste statique de nœuds qu'il connaît. Il va alors essayer de contacter ces nœuds et éventuellement joindre le réseau ;
- *Un serveur central.* Cette stratégie peut être utilisée lorsque le réseau est un réseau hybride et qu'un serveur central existe. Chaque nœud connaît alors l'adresse de ce serveur. Lorsqu'ils veulent se connecter, les nœuds s'enregistrent auprès de ce serveur qui leur fournit la liste des nœuds déjà inscrits ;
- *Une diffusion locale.* Avec cette stratégie, chaque nœud qui veut se joindre au réseau va diffuser un message. Ainsi, lorsqu'un nœud du réseau reçoit ce message, il va répondre et une communication est alors établie avec le nœud qui veut se joindre ;
- « *Buddy list* ». Cette stratégie s'apparente à la première, à l'exception que la liste n'est pas statique. En effet, cette liste va contenir les nœuds déjà vus ainsi que les nœuds favoris. Lorsque le nœud se reconnecte au réseau, il peut mettre à jour sa liste avec les nouvelles informations.

L'adressage des nœuds Afin de pouvoir localiser les nœuds et de les distinguer, il faut absolument adopter une stratégie d'adressage des nœuds. Plusieurs stratégies

sont envisageables :

- *Un adressage statique.* Dans cette stratégie, chaque nœud possède une configuration statique explicite des autres nœuds du réseau. On parle alors d'un réseau P2P statique. Un exemple est l'utilisation des fichiers « hosts », présents dans tous les systèmes d'exploitation modernes ;
- *Un adressage centralisé.* Cette stratégie est utilisable dans un réseau hybride où un serveur central existe. Ce serveur va alors se charger de distribuer les adresses aux nœuds qui s'enregistrent. Ce mécanisme s'apparente au mécanisme DHCP, souvent utilisé dans les réseaux locaux ;
- *Un adressage basé sur la découverte de voisins.* Lorsqu'un nœud se connecte au réseau P2P, il va utiliser ses voisins pour se connecter aux services. Ainsi, il suffit que chaque nœud puisse identifier ses voisins. Lorsqu'une requête est lancée, elle est propagée à travers le réseau. La réponse va tout simplement suivre le chemin inverse. Cette stratégie est particulièrement intéressante pour les réseaux de partage de fichiers.

2.12 Applications des réseaux P2P

Il existe trois grandes catégories d'applications pour les réseaux P2P, soit l'informatique collaborative, la messagerie instantanée ainsi que les communautés d'affinité. L'informatique collaborative, appelée également informatique répartie (*distributed computing*), regroupe la puissance de calcul et/ou l'espace libre sur les pairs du réseau. Ce type de réseau est très populaire chez les organisations scientifiques et biotechniques qui ont besoin de grandes ressources de calcul. Un exemple d'une telle application est le projet SETI@HOME. D'autres exemples peuvent être trouvés sur grid.org qui héberge une multitude de projets d'informatique collaborative. Ce réseau comporte environ 2 millions d'ordinateurs personnels et génère environ 100 teraflops de puissance.

Une forme très commune des réseaux P2P est la messagerie instantanée (MI) où des applications permettent à des usagers de communiquer via des messages textes en temps réel. Alors que la plupart offrent une version gratuite de leur logiciel MI, certaines compagnies ont commencées à se concentrer sur des versions pour entreprises, puisqu'elles utilisent de plus en plus la MI comme outil de communication standard. La messagerie instantanée a gagné énormément en popularité au cours des dernières

années. Plusieurs exemples de MI existent pour les réseaux filaires, tel Internet. Dans les logiciels gratuits, on retrouve notamment MSN, ICQ, Yahoo et AOL. Dans les logiciels commerciaux, on retrouve notamment JabberNow.

Les communautés d'affinité ont été popularisées par les groupes de réseaux P2P qui sont basés autour de partage de fichiers, tel que Kazaa, Napster et Gnutella. Cependant, le partage de fichier n'est pas la seule application des communautés d'affinité. En effet, ces communautés, qui peuvent être publiques ou privées, sont des groupes d'individus travaillant sur une application commune. Deux applications bien connues dans le monde informatique sont Microsoft NetMeeting et Groove. Ces applications permettent à un groupe de personnes d'avoir un espace de travail virtuel commun sans nécessairement que leur emplacement le soit. Ils permettent donc de partager des applications, de démarrer des conférences et, bien-sûr, de partager des fichiers.

2.13 Protocoles P2P de partage de fichiers

Les protocoles P2P de partage de fichiers les plus matures ont été développés pour des réseaux filaires tels qu'Internet. Ils peuvent être classifiés en trois générations.

Première génération La première génération des réseaux P2P de partage de fichiers se base sur des réseaux P2P hybrides. Il existe donc un serveur centralisé qui assure le bon fonctionnement du réseau. L'étendu de la responsabilité du serveur dépend du protocole. L'application P2P de première génération la plus connue est Napster. La première version de Napster fut développée en juin 1999 par Shawn Fanning, avec l'aide volontaire de Sean Parker. Fanning voulait développer une méthode pour trouver de la musique plus simple d'utilisation que les moteurs de recherche. Napster se base sur un réseau P2P hybride avec un serveur de découverte et de requête. À la suite d'une injonction de la cours, le réseau Napster a du cesser ses opération en juin 2001. Pour détruire le réseau, il a suffit de détruire le serveur central. Les pairs ne pouvaient donc plus se retrouver ni effectuer des requêtes.

Deuxième génération Suivant les problèmes légaux de Napster, Justin Frankel de Nullsoft a décidé de créer un réseau P2P sans serveur centralisé et le résultat fut Gnutella. La première version de ce réseau considérait tous les pairs égaux entre eux,

ce qui a engendré d'énormes congestions avec l'arrivée massive d'utilisateurs. C'est alors que le protocole FastTrack (utilisé notamment par Kazaa) a été développé. Celui-ci crée une hiérarchie entre les pairs. Ceux qui ont davantage de capacité sont élus nœuds d'indexage. Cela permet d'avoir une meilleure évolutivité du réseau. Gnutella a adopté la même philosophie pour créer une nouvelle version plus performante. Gnutella et FastTrack sont deux protocoles qui implémentent des réseaux de recouvrement non-structurés. La seconde génération a également vu naître les protocoles qui utilisent les DHT, donc les protocoles avec réseaux de recouvrement structurés. Les protocoles les plus connus incluent CAN, Chord, Pastry et Tapestry (et sa nouvelle implémentation Chimera). Un autre protocole intéressant issu de la deuxième génération est BitTorrent. Certaines de ses caractéristiques ont inspiré la solution proposée dans ce mémoire. BitTorrent est un protocole de partage de fichiers qui utilise la bande passante des pairs qui téléchargent. Lorsqu'un fichier est mis sur un serveur HTTP, ce dernier doit assumer toute la bande passante requise pour transférer aux nœuds qui télécharge. Ainsi, plus la demande est grande, plus le serveur est taxé. Dans un système de partage de fichier P2P traditionnel, le fichier est desservi par tous les nœuds qui le possède, donc la charge est répartie. BitTorrent pousse ce concept encore plus loin en permettant aux nœuds qui téléchargent de partager les données qu'ils possèdent, même s'ils n'ont pas encore le fichier en entier. Pour réaliser cette technique, chaque fichier est divisé en bloc et les nœuds publient alors la liste de blocs qu'ils possèdent. Ainsi, des nœuds peuvent se partager des blocs sans qu'aucun ne possède le fichier dans sa totalité. Les avantages sont importants :

- *une meilleure performance et évolutivité.* Puisque la bande passante de tous les pairs est utilisée, le réseau peut supporter, théoriquement, un nombre infini de pairs, puisque chacun contribue au partage ;
- *une plus grande robustesse.* Même si tous les pairs qui possèdent le fichier en entier disparaissent, les autres nœuds peuvent continuer à s'échanger des blocs. De plus, si chaque bloc se retrouve au moins une fois dans le réseau, le fichier peut être reconstitué au complet.

BitTorrent est un protocole très versatile. En effet, il peut être utilisé avec un réseau de recouvrement non-structuré ou encore un réseau structuré avec l'aide de DHT. Dans le mode non-structuré, il utilise un serveur central (Tracker) qui sert de serveur de découverte.

Troisième génération La troisième génération de réseaux P2P est axée sur la confidentialité. Les protocoles de cette génération incorporent des mécanismes d'encryption qui permettent de masquer l'information échangée. De plus, ces protocoles vont acheminer les paquets suivants des routes préétablies. Des exemples de protocoles connus sont Freenet, I2P, GNUnet et Entropy. Cependant, ces protocoles n'ont pas gagné en popularité notamment à cause d'un surdébit énorme ajouté par la confidentialité.

2.14 Différences entre P2P sur Internet et P2P sur MANETs

Il existe plusieurs différences fondamentales entre le déploiement d'un réseau P2P sur un réseau filaire tel Internet et sur un réseau mobile sans fil ad hoc :

- *La limitation de la bande passante.* La capacité réseau inférieure des liens radio réduit la rentabilité des protocoles P2P sur les réseaux ad hoc, notamment ceux qui présentent un surdébit énorme ;
- *Les interférences causées par les accès multiples.* Dans des réseaux sans fil, il est nécessaire d'utiliser des techniques d'accès multiples tel CSMA/CA pour obtenir le canal sans fil et transmettre de l'information. Comme il n'existe aucun point de coordination central dans les réseaux ad hoc, il y a souvent collisions et des délais pour l'obtention du canal. Ces problèmes peuvent empirer avec l'utilisation d'applications P2P qui utilisent de messages générant de forts surdébits ;
- *La forte mobilité des nœuds.* Sur Internet, la topologie d'un réseau de recouvrement P2P change sur de grandes échelles de temps. Dans un réseau ad hoc, les transmissions limitées et la mobilité des nœuds provoquent des changements fréquents à la topologie. Cela force donc les applications P2P s'exécutant sur des réseaux ad hoc à mettre à jour plus fréquemment la topologie du réseau de recouvrement afin de maintenir la cohésion entre ce dernier et le réseau physique ;
- *L'ajout et la suppression fréquente de nœuds (churn).* Les réseaux de recouvrement P2P structurés sont particulièrement affectés par ce phénomène. Sur Internet, ce phénomène se produit car les nœuds sont généralement des ordinateurs de bureaux qui ne sont pas nécessairement en marche tout le temps.

Dans un réseau ad hoc, en plus d'avoir la notion de marche/arrêt, les nœuds peuvent également perdre contact avec les autres nœuds à cause de la mobilité, accentuant ainsi le problème ;

- *Le manque d'infrastructure.* Certains protocoles P2P utilisent des composantes d'infrastructure dans leur conception. Par exemple, un protocole de routage P2P peut assigner des identificateurs aux nœuds basés sur leur emplacement et déterminés à partir de points de repère statiques afin d'améliorer le routage. Ces techniques sont difficiles à implémenter dans les réseaux ad hoc ;
- *Une énergie limitée.* La plupart des applications P2P sur Internet ne sont pas conçues pour opérer avec un minimum de transmission de messages. Dans le cas de réseaux ad hoc, il est très important de réduire le nombre de transmissions, tout en gardant une performance acceptable ;
- *L'adressage des nœuds.* Les nœuds dans un réseau ad hoc sont plus susceptibles de se déconnecter et de se reconnecter au réseau. Même s'il n'existe pas d'architecture normalisée pour l'adressage dans les réseaux ad hoc, il est plausible de présumer que les nœuds vont avoir des adresses différentes dans le temps. Pour les réseaux structurés, cela peut devenir problématique quand vient le temps de faire l'association entre l'adresse logique et l'adresse physique (nodeID-à-IP).

2.15 Protocoles de partage de fichiers pour MANETs

Dans cette section, nous présentons une revue sélective de différents protocoles de partage de fichier conçus spécialement pour les réseaux mobiles ad hoc.

2.15.1 Adaptation des protocoles filaires déjà existants

Oliveira et al. (Oliveira *et al.*, 2003) ont étudié la performance d'une application P2P (basée sur Gnutella) non structurée dans un réseau ad hoc avec trois protocoles de routage, soit DSR, AODV et DSDV. Leur travail montre qu'une telle application performe beaucoup moins bien qu'une simple application unicast. La raison principale est que comme le réseau de recouvrement est non structuré, les relations de voisinage ne sont pas transmises du réseau original au réseau de recouvrement. Ainsi, deux nœuds voisins dans le réseau de recouvrement peuvent être séparés de plusieurs sauts

dans le réseau physique. Le résultat de plusieurs requêtes peut rapidement saturer le réseau ad hoc.

2.15.2 Diffusion épidémique d'information

Les premiers protocoles de partage de fichiers proposés pour les réseaux ad hoc sont basés sur la notion de diffusion épidémique d'information. Cette technique permet la diffusion d'information dans un réseau à la manière d'une infection contagieuse : chaque nœud du réseau qui reçoit de l'information va la transmettre aléatoirement à certains de ses voisins qui vont effectuer le même processus jusqu'à ce que l'information soit propagée à tous les nœuds du réseau.

La première implémentation, intitulée 7 Degrees of Separation (7DS), a été proposée par Papadopouli et Sidiroglo (Papadopouli et Schulzrinne, 2000). Sont but est de permettre aux nœuds de partager des documents Web, facilitant ainsi la navigation en ligne sans être connecté à Internet. Lindermann et Waldhorst (Lindemann et Waldhorst, 2002) ont proposé un autre système basé sur la diffusion épidémique intitulé Passive Distributed Indexing (PDI). Il s'agit d'une méthode générale de recherche de paires « clef, valeur ». Cette technique est conçue spécialement pour les réseaux mobiles car elle profite justement de la mobilité des nœuds et des mécanismes de diffusion qu'offrent les réseaux sans-fil pour effectuer la diffusion d'information. Chaque nœud contient une base de données des pairs « clef, valeur ». Lorsqu'un nœud recherche une ressource, il en fait la demande à ses voisins. Une requête consiste à trouver la valeur associée à une clef. Chaque nœud qui reçoit une requête la transmet ensuite à ses voisins immédiats et ainsi de suite. De plus, chaque nœud écoute les réponses envoyées et met à jour sa base de données avec les nouvelles paires entendues. Si un nœud se déplace dans une autre région du réseau, il va amener avec lui sa base de données contenant les paires qu'il connaît. Ainsi, les requêtes populaires risquent fortement de s'y retrouver et seront transmises à la nouvelle région.

2.15.3 ORION

ORION (Klemm *et al.*, 2003) est un système de partage de fichiers P2P spécialement conçu pour les réseaux ad hoc. Il consiste en un algorithme de construction et de maintenance d'un réseau de recouvrement au niveau de la couche application qui permet l'acheminement des messages essentiels au fonctionnement du système

(requêtes, réponse, transmissions de fichier). Les liens du réseau de recouvrement sont construits sur demande, maintenus uniquement pendant leur utilisation et suivent les liens physiques du réseau de base. ORION combine le processus de requête au niveau de la couche application avec le processus de découverte de route au niveau de la couche réseau, réduisant ainsi les surcharges de contrôle et augmentant l'efficacité des recherches par rapport à un algorithme non spécialisé qui s'exécute au dessus du protocole de routage. De plus, le réseau de recouvrement permet un faible taux de surdébit pour les transferts de fichiers ainsi qu'une probabilité de succès plus élevée.

Il est intéressant de noter que le fonctionnement d'ORION ne dépend pas de l'algorithme de routage utilisé. Cependant, comme certains processus employés par ORION mimique ceux utilisés par certains protocoles (notamment les protocoles réactifs), il est intéressant d'avoir, dans ce cas-ci, une approche intégrée où la couche application et la couche réseau collaborent dans le but de ne pas dupliquer les efforts.

ORION consiste en deux grandes parties, soit l'algorithme de recherche et le protocole de transfert.

L'algorithme de recherche ORION implémente un algorithme de recherche efficace qui permet d'effectuer une recherche de fichiers à partir de mots clés. Cet algorithme combine le processus de recherche de la couche application au processus de découverte de routes de la couche réseau. Il combine des techniques empruntées à AODV ainsi qu'au protocole Simple Multicast Broadcast (SMBP).

Chaque nœud mobile maintient un répertoire local des fichiers à partager. Il est supposé qu'un identificateur unique est associé à chaque fichier. De plus, deux tables de routage sont maintenues : une table de réponses et une table de routage de fichiers. Comme dans le protocole AODV, les tables de réponses servent à garder en mémoire le nœud à partir duquel une requête est parvenue comme prochain saut dans le chemin inverse. Ainsi, un nœud peut acheminer les réponses sans devoir explicitement initier une nouvelle découverte. La table de routage de fichiers est une structure de données qui garde les prochains sauts optionnels pour les transferts de fichiers basés sur l'identificateur de fichier. ORION met à jour les deux tables durant le processus de requête. La consommation de mémoire est contrôlée en limitant leurs tailles et en appliquant la politique de remplacement Least Recently used.

Pour la phase de découverte, il existe deux types de messages, soit les messages requête (QUERY) et les messages réponses (RESPONSE). Un message requête

contient une chaîne de caractères qui consiste en un ou plusieurs mots-clefs. Un message réponse contient un ou plusieurs identificateurs uniques correspondants à un ou plusieurs fichiers qui satisfont la requête. De plus, chaque message contient l'identificateur du nœud source d'où il provient ainsi qu'un numéro de séquence qui est strictement ascendant. Le débordement du numéro de séquence est contrôlé de la même manière que dans AODV. Ces deux champs restent inchangés durant le transfert du message. Ainsi, en gardant en mémoire le plus grand numéro de séquence pour chaque nœud, un nœud peut éviter de relayer un même message plusieurs fois.

Un message requête est diffusé au niveau de la couche liaison (des données de l'application sont encapsulés dans des messages broadcast de la couche liaison). Cette technique est utilisée dans le protocole SMBP. De plus, lors de son acheminement, le message requête va permettre d'établir le chemin inverse dans les tables de routage des nœuds intermédiaires. Cette technique est également utilisée dans le protocole AODV.

Lorsqu'un nœud reçoit un message de requête, il vérifie s'il possède des fichiers qui satisfont la requête. Si c'est le cas, il retourne alors un message réponse avec les identificateurs des fichiers au nœud source. De plus, il achemine l'original du message requête à ses voisins. S'il reçoit une réponse d'un de ses voisins, avant de l'acheminer à la source, il vérifie s'il possède ce fichier. Si c'est le cas, il ignore la réponse. Sinon, il ajoute une entrée dans sa table de routage de fichiers indiquant que le nœud est le prochain saut vers le fichier. Cependant, si cette entrée n'est pas unique, la réponse est ignorée. Sinon, elle est acheminée. Ainsi, lorsqu'un nœud exécute une recherche, il ne recevra, au maximum, qu'une seule source pour chaque fichier unique.

Le protocole de transfert Au démarrer, le protocole de transfert utilise les routes trouvées dans les tables pour la signalisation et la transmission de données. La table de routage de fichiers peut contenir plusieurs chemins vers un même fichier. À cause de la nature des réseaux ad hoc, le nœud source d'un fichier peut changer à tout moment. C'est pour cette raison qu'ORION donne le contrôle complet du transfert au nœud destination. Le fichier est divisé en blocs de même taille. La destination envoie une requête de donnée, DATA_REQUEST, pour un bloc sur le chemin trouvé dans sa table de routage de fichiers. Lorsque ce message est reçu par un nœud possédant le fichier, celui-ci répond par un DATA_REPLY, contenant le bloc en question. Ce message emprunte le chemin inverse de la requête. Une fois le message réponse reçu,

le nœud renvoie une requête pour un prochain bloc et le cycle recommence.

ORION achemine la signalisation et les données sur le meilleur lien parmi tous les liens que le nœud connaît. Lorsqu'une route est brisée, ORION effectue une résolution locale, contrairement aux protocoles de routage réactifs comme AODV ou DSR. Ainsi, lorsqu'une route est brisée, au lieu d'avertir le nœud source afin qu'il recommence une recherche, c'est le nœud intermédiaire qui réalise que la route est brisée et qui renvoie le message sur une route alternative se trouvant dans sa table. Dans le cas où aucune route alternative n'est disponible, le nœud intermédiaire avertit son précédent qui retransmet le message sur la route alternative qu'il possède dans sa table. Si tous les nœuds intermédiaires ne possèdent par de routes alternatives dan leur table, le dernier nœud intermédiaire va alors avertir son précédent, soit le nœud source du message. Si celui-ci ne possède également pas de route, il a alors le choix d'annuler le transfert ou d'effectuer la requête à nouveau. S'il décide de refaire la requête, celle-ci se fera directement sur l'identificateur du fichier.

Pour augmenter la performance, il est important qu'ORION soit avertit au plus vite lorsqu'un lien est brisé. Si c'est possible, ORION sollicite la coopération de la couche réseau pour obtenir cette information. Sinon, il faut accuser chaque message envoyé.

Si un paquet DATA_REQUEST ou DATA_REPLY est perdu, le protocole de transfert reste bloqué en attente du paquet perdu. Pour palier à ce problème, ORION incorpore un ordonnancement de paquets et un mécanisme de recouvrement en cas de perte. Lorsque le nœud qui reçoit le fichier n'a pas eu de réponse après un certain délai, il va envoyer une nouvelle requête pour le prochain bloc. Dans un réseau P2P, l'ordre d'arrivée des paquets est peu pertinent. Ce qui est important c'est que chaque paquet soit reçu au moins une fois. Ainsi, ORION ne redemandera pas un paquet avant d'avoir fini un cycle. Le principal avantage de cette méthode est qu'elle alloue du temps supplémentaire au paquet dans le cas où sa transmission est plus longue. Si le cycle est terminé et que le paquet n'est toujours pas reçu, le nœud va redemander une nouvelle transmission du paquet.

2.15.4 BTM

Rajagopalan et Shen (Rajagopalan et Shen, 2006) on proposé une adaptation du protocole BitTorrent pour les réseaux ad hoc intitulée BTM. Le protocole traite

les problèmes rencontrés lorsque le protocole BitTorrent est déployé tel quel sur un réseau ad hoc. Le protocole traite notamment le problème de centralisation. Ainsi, à la place d'avoir un serveur de découverte central (Tracker), BTM utilise une approche décentralisée pour la découverte des pairs. L'approche consiste en une coopération inter-couche avec ANSI comme protocole de routage. De plus, BTM offre un mécanisme de réplication. Ainsi, pour chaque nœud qui possède un fichier à partager, on retrouve des nœuds proxy qui vont immédiatement télécharger le fichier et le desservir. Les simulations effectuées par les auteurs concluent que BTM est plus performant qu'un simple déploiement de BitTorrent sur les réseaux ad hoc.

CHAPITRE 3

STRATÉGIE DE PARTAGE DE FICHIERS PROPOSÉE

Le Chapitre 1 a permis de fixer les objectifs de recherche qui devront être satisfaits dans le présent chapitre. Dans le Chapitre 2, nous avons passé en revue les différents mécanismes reliés au routage dans les réseaux ad hoc ainsi que les principaux types d'algorithmes et implémentation. Ensuite, nous avons passé en revue les différents mécanismes nécessaires à l'implémentation d'un réseau de partage P2P ainsi que les principales implémentations disponibles. Nous énoncerons dans ce chapitre la solution que nous proposons au problème de partage de fichiers P2P sur les réseaux mobile ad hoc.

Comme toute autre solution de partage de fichiers P2P, la solution proposée incorpore les deux algorithmes requis au fonctionnement, soit un algorithme de recherche et un algorithme de transfert. L'algorithme de recherche va permettre à tout nœud désirant trouver un fichier de localiser les nœuds qui desservent ce fichier. Une fois la localisation est réussie, l'algorithme de transfert peut être démarré permettant ainsi au nœud de recevoir le fichier voulu.

3.1 Hypothèses

Dans cette partie sont exposées les hypothèses retenues lors du développement de la solution :

- On suppose donc un réseau sans fil composé de nœuds connectés de manière ad hoc. Les nœuds du réseau peuvent être mobiles.
- Sur ce réseau est déployé un protocole de routage quelconque. La solution proposée fonctionne indépendamment du protocole de routage choisi. Nous verrons plus tard qu'il y a possibilité d'utiliser un protocole adapté à notre solution.
- Chaque nœud possède une banque de fichiers qu'il désire partager avec ses pairs.

- Un nœud peut à tout moment effectuer une recherche ou démarrer un transfert.
- On suppose qu'aucun nœud malicieux n'existe dans le réseau au niveau du routage des paquets. Ainsi, tous les nœuds participent positivement au bon fonctionnement du réseau. Si la possibilité qu'un nœud malicieux s'y retrouve est réelle, un mécanisme de détection doit être déployé. Des nœuds malicieux peuvent cependant exister s'ils desservent de fausses données lors d'un transfert. Ce scénario sera traité plus tard dans le chapitre.

3.2 Description sommaire de la solution

La solution proposée s'inspire du protocole ORION. L'objectif est que la solution proposée offre de meilleures performances de transfert par rapport à ORION. Principalement, elle se base sur une technique introduite par BitTorrent où chaque client qui télécharge un fichier soit également une source partielle de ce fichier, desservant d'autres clients. Cette technique adaptée aux réseaux mobiles ad hoc permet de réduire le nombre de transmissions requises pour effectuer le transfert. Ceci se traduit par une réduction de la bande passante utilisée, de la diminution des collisions et congestion, et surtout de la diminution du temps requis pour compléter les téléchargements.

La solution proposée permet également une meilleure gestion des erreurs de transferts. Contrairement à ORION qui ne vérifie qu'une fois le transfert terminée, la vérification de la validité des données se fait morceau par morceau. Ceci réduit la taille des données retransmises lors d'erreurs.

3.3 Banque de données

La banque de données d'un nœud peut contenir zéro, un ou une infinité de fichiers. La seule contrainte est au niveau de la mémoire du mobile, donc une contrainte physique. Chaque fichier possède un nom, une taille et un contenu (les données). Chaque fichier est divisé en morceaux de données de tailles égales (sauf le dernier, qui peut être de taille inférieure). La taille d'un morceau doit être choisie de sorte à maximiser l'efficacité du protocole. Une taille trop grande réduit le nombre de sources disponibles pour un client, alors qu'une taille trop petite augmente le débit

requis pour la signalisation. Il est suggéré d'utiliser une taille qui est une puissance de 2 se trouvant entre 32 Ko et 256 Ko.

Chaque morceau d'un fichier possède sa propre signature. La signature est le résultat d'une fonction de hachage appliquée aux données contenues dans le morceau. L'algorithme de hachage suggéré est l'algorithme MD5 [1], mais tout autre algorithme est envisageable (SHA1 [2] par exemple). Le fichier possède également une signature. Contrairement aux morceaux, la signature d'un fichier n'est pas le résultat du hachage de son contenu. La signature d'un fichier est plutôt le résultat du hachage de la chaîne obtenue en concaténant les signatures de tous ses blocs, dans l'ordre. Cette méthode de calcul de la signature du fichier permet une meilleure sécurité, mécanisme exposé plus tard dans le chapitre. Deux fichiers sont considérés identiques s'ils possèdent la même signature, donc les mêmes signatures de blocs et ce, peu importe leur nom.

Les signatures des blocs et des fichiers doivent être disponibles en tout temps à l'application. Deux stratégies sont envisageables :

- Les signatures sont calculées lorsque l'application démarre pour tous les fichiers dans la banque de données. Cette stratégie est couteuse en termes de temps de calcul du processeur et peut, dans le cas d'une grande base de données, réduire le niveau de la batterie du mobile.
- Les signatures sont enregistrées sur la mémoire non volatile dans un fichier et lu lorsque l'application démarre. Cette stratégie a l'avantage de ne pas surcharger le processeur mais requiert cependant de l'espace additionnel sur la mémoire non volatile. Comme la mémoire non volatile est généralement assez grande, puisqu'elle contient les fichiers, et que l'espace requis pour stocker les signatures des fichiers est plutôt négligeable par rapport aux fichiers, la deuxième stratégie est, dans la plupart des cas, recommandée.

3.4 Tables

Pour son fonctionnement, le protocole proposé maintient deux tables, soit une table de routage et une table de transfert.

La table de routage sert à stocker les routes. Ainsi, pour une destination donnée, on y retrouve une liste les prochains sauts pour l'atteindre. Cette table est similaire à la table utilisée par le protocole réactif AODV. Il s'agit donc une table de correspondance. Le Tableau 3.1 montre les champs qui composent une entrée de cette table.

Ainsi, pour chaque destination, on retrouve :

- *Une liste ordonnée de couples* « prochain saut, nombre de sauts », par ordre croissant de nombre de sauts. La liste peut être limitée afin de réduire l'occupation de la mémoire.
- *Le dernier numéro de séquence rencontré.* Il s'agit d'un entier non signé qui correspond au plus grand numéro de séquence rencontré pour ce nœud. Son utilisation est expliquée plus tard dans ce chapitre.
- *Un timestamp.* Il s'agit d'un entier non signé correspondant au dernier temps de rafraichissement de la destination. Ce champ est utile si la taille la table doit être limitée. Dans ce cas, le remplacement ce fait suivant la règle du Least Recently Used (LRU).

Pour accéder à une entrée dans cette table, il faut lui fournir l'adresse du nœud en question.

La table de transfert permet de garder toutes les informations pertinentes par rapport aux fichiers en cours de transfert ou aux fichiers qui ont été recherchés (et qui risquent donc d'être transférés). Le Tableau 3.2 montre les champs qui composent une entrée de cette table. On retrouve donc, pour un transfert donné, les champs suivants :

- *Le nom.* Il s'agit d'une chaîne de caractères de taille variable.
- *La taille du fichier.* Il s'agit d'un entier non signé (6 octets).
- *Le bitfield.* Il s'agit d'une chaîne de bit qui indique l'état de tous les morceaux du fichier. La taille de ce champ est variable et dépend du nombre de morceaux. Son calcul est le suivant : ceil(taille du fichier/taille d'un morceau/8). Chaque bit correspond à l'état d'un morceau, dans l'ordre. Par exemple, si le bit 7 est mis à 1, le morceau 7 est présent. S'il est mis à 0, le morceau est absent.
- *Le nombre de morceaux possédés.* Il s'agit d'un entier non signé indiquant le nombre de morceaux que le nœud possède. Cette information est redondante mais évite le calcul à partir du bitfield, opération qui est couteuse.

TABLEAU 3.1 Entrée dans la table de routage

Champ	**Taille en octets**
Liste ordonnée de couples	Variable
Dernier numéro de séquence	4
Timestamp	4

- *La liste de sources.* Il s'agit d'une liste d'adresses des nœuds qui possèdent le fichier. Cette liste peut être limitée en mémoire. Le Tableau 3.3 montre les champs qui composent une entrée dans cette liste. On retrouve l'adresse de la source et un drapeau indiquant l'état de la source. Si le drapeau est mis à vrai, la source est complète. Sinon, la source est partielle et dans ce cas, un troisième champ est rajouté, soit le bitfield de la source pour ce fichier.

Pour accéder à une entrée dans cette table, il faut lui fournir la signature du fichier en question.

3.5 Algorithme de recherche

Dans cette partie, les détails techniques de l'algorithme de recherche sont exposés. La Figure 3.1 montre les étapes majeures de cet algorithme. On appelle nœud requête le nœud qui recherche un fichier. On appelle nœud réponse tout nœud qui possède ce fichier.

Il existe deux types de recherche possible. Le premier type de recherche pour la solution proposée est une recherche basée sur mots-clefs. Il s'agit du type de recherche le plus répandu et ce, non seulement dans les systèmes de partage de fichiers (engins de recherches, par exemple). Lorsqu'un usager désire rechercher un fichier, il entre un ou plusieurs mots clefs. Ces mots clefs servent comme base de comparaison, pour les autres nœuds, avec les noms de tous leurs fichiers. La recherche peut être simple. Par exemple, l'usager pourrait désirer trouver tous les fichiers contenant le mot « MANET ». Il va donc lancer une recherche avec comme mot clef « MANET ». L'usager peut également effectuer des recherches plus complexes. Lorsqu'un usager effectue une recherche, il a l'option de préciser les paramètres suivants :

- *Un ou plusieurs mots clefs.* Un mot clef va être comparé avec tous les mots

TABLEAU 3.2 Entrée dans la table de transferts

Champ	**Taille en octets**
Nom du fichier	Variable
Taille du fichier	6
Bitfield	Ceil(Taille Fichier / Taille Paquet / 8)
Nb morceaux possédés	4
Liste des sources	Variable * 4 (IPv4)

TABLEAU 3.3 Entrée dans la liste de source de la table de transferts

Champ	Taille en octets
Adresse	4 (IPv4)
État	1
Bitfield (optionnel)	Dépend du nnombre de morceaux

contenus dans un fichier. Chaque mot clef peut être précédé par le symbole -, qui indique que le mot ne doit pas figurer dans le nom des fichiers recherchés ;

- *Une phrase exacte.* Les phrases sont contenues à l'intérieur de guillemets (« et »). La différence entre une phrase exacte et une série de mots clefs est l'importance de l'ordre. Une phrase exacte est trouvée si et seulement si tous les mots clefs qu'elle contient s'y retrouvent dans le même ordre que dans la phrase. Comme pour les mots clefs, une phrase peut être précédée par le symbole -, qui indique que la phrase ne doit pas figurer dans le nom des fichiers recherchés. Il est à noter que les mots contenus dans la phrase peuvent, quant à eux, peuvent s'y retrouver s'ils ne sont pas dans le même ordre que la phrase ;
- *Des contraintes sur la taille du fichier.* L'usager peut spécifier la taille minimale du fichier, la taille maximale du fichier ou même la taille exacte du fichier (ce qui se traduit par des tailles minimale et maximale égales).

L'autre type de recherche possible est une recherche basée sur la signature du fichier. L'usager peut donc spécifier la signature du fichier qu'il désire télécharger. Ainsi, ce type de recherche ne peut retourner qu'un seul fichier, soit celui qui possède cette signature. Ce type de recherche n'inclut aucun autre critère.

Une fois que l'usager entre ses paramètres de recherche, le processus de recherche du fichier débute. La première étape est la construction d'un message FREQ. Il existe deux types de messages FREQ, soit FREQK pour une requête par mots clefs et FREQS pour une requête par signature. Tout message FREQ contient, en plus des champs génériques, les champs suivants :

- *Un numéro de séquence.* Il s'agit d'un entier, non signé, qui identifie la requête. Pour chaque nouvelle requête, le nœud doit inscrire un numéro unique (un incrément du numéro précédent). La paire « adresse source, numéro séquence » permet d'une indentification unique pour chaque requête. Le numéro de séquence démarre à 0. Lorsque la valeur maximale est atteinte, il redémarre à 0.

- *Le nombre de sauts.* Il s'agit d'un octet qui permet d'avoir une mesure de l'efficacité de la route. À chaque saut, il est incrémenté par le nœud intermédiaire. Un message FREQK étend le message FREQ et est montré au Tableau 3.4. Les champs propres à un message FREQK sont les suivants :
- *La taille minimale.* Il s'agit d'un entier non signé qui représente la taille minimale que le fichier peut avoir. S'il n'est pas précisé, la valeur est 0.
- *La taille maximale.* Il s'agit d'un entier non signé qui représente la taille maximale que le fichier peut avoir. S'il n'est pas précisé, la valeur est 0.
- *La taille de la requête.* Il s'agit d'un nombre non signé qui représente le nombre de caractères qui forment la requête.
- *La requête elle-même (les critères).* Ce champ est de taille variable et contient tous les critères que l'usager a spécifiés. Les phrases sont entourées par des guillemets.

Un message FREQS étend le message FREQ et est montré au Tableau 3.5. L'unique champ propre à un message FREQS est la signature du fichier. Il s'agit d'une chaîne d'octets correspondant à la signature du fichier recherché. La taille de ce champ est fixe et dépend de la fonction de hachage utilisée (16 octets pour MD5).

Lorsqu'un message FREQ est construit, le numéro de séquence inscrit dans le message doit être un nouveau numéro (l'ancien numéro + 1). Le dernier saut est mis à son adresse. Le nombre de sauts est initialisé à 1. Le TTL est initialisé au nombre de sauts maximal permis pour la requête. Afin de ne pas bombarder tout le réseau avec une requête, une approche progressive peut être utilisée, comme le font certains protocoles comme AODV. Ainsi, le nœud débute avec un TTL petit. S'il ne reçoit pas réponse dans un certain délai, il renvoie le message avec un TTL plus grand. Il est à noter qu'il doit également incrémenter le numéro de séquence, sinon la requête sera ignorée par ses voisins.

Une fois le message FREQ construit, il est transmis par une diffusion générale à tous ses voisins. Lorsqu'un nœud reçoit un message FREQ, il va trouver dans sa table de routage le dernier numéro de séquence pour la source du message. Si le numéro de séquence dans la table est égal ou supérieur, la requête a déjà été traitée et elle est ignorée. Si le numéro de séquence dans la table est inférieure ou s'il n'existe pas d'entrée dans la table de routage pour la source, la requête est nouvelle et doit être traitée. Dans ce cas, le nœud débute par la diffusion du message à ses voisins, en s'assurant d'incrémenter le nombre de sauts et en modifiant le champ Dernier Saut

TABLEAU 3.4 Message FREQK

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Numéro de séquence	4
Nombre de sauts	1
Taille minimale	4
Taille maximale	4
Taille requête	4
Critères	Variable

TABLEAU 3.5 Message FREQS

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Numéro de séquence	4
Nombre de sauts	1
Signature	16 (MD5)

en inscrivant son adresse. Ensuite, il met à jour sa table de routage en inscrivant le numéro de séquence et en rajoutant la paire « dernier saut, nombre de sauts » à la liste de l'entrée correspondant à l'adresse de la source. Dans le cas où la table de routage ne possède pas une d'entrée pour l'adresse de la source, elle doit être créée. Finalement, le nœud doit consulter sa base de fichiers afin de vérifier s'il possède au moins un fichier qui satisfait les critères. Si un tel fichier n'existe pas, le traitement du message FREQ est terminé. S'il existe au moins un fichier, le processus de réponse doit être entamé. On suppose que le nombre maximal de fichiers pouvant satisfaire une requête est de 255. S'il existe plus de 255 fichiers, seuls les premiers 255 sont retenus.

Le processus de réponse à une requête de fichier débute par la construction d'un message FREP, qui sera envoyé à la source du message FREQ correspondant. Contrairement au message FREQ, le message FREP est unicast. Le Tableau 3.6 montre la structure d'un tel message. Le message FREP étend le message générique en ajoutant les champs suivants :

- *Numéro de séquence.* Il s'agit du même champ que celui dans le message REQ. Il est utilisé par la source afin d'identifier la requête pour laquelle correspond cette réponse ;
- *Le nombre de fichiers.* Il s'agit d'un octet non signé indiquant le nombre de fichiers respectant les critères. Ce nombre est donc limité à 255 ;
- *Liste des fichiers.* Il s'agit d'une liste de taille variable décrivant les fichiers respectant les critères. Le Tableau 3.7 montre les champs qui composent une entré dans cette liste. On y retrouve donc les champs suivants :
  - *La taille du nom du fichier.* Il s'agit d'un entier non signé qui contient la taille du nom du fichier ;
  - *Le nom du fichier.* Il s'agit d'une chaîne de caractères variable qui contient le nom du fichier ;
  - *La taille du fichier.* Il s'agit d'un entier (6 octets) indiquant la taille du fichier, pour une taille maximale de 256 téraoctets ;
  - *La signature du fichier.* Il s'agit d'une chaîne d'octets correspondant à la signature du fichier recherché. La taille de ce champ est fixe et dépend de la fonction de hachage utilisée (16 octets pour MD5) ;
  - *L'état du fichier.* Il s'agit d'un drapeau qui indique si le fichier est complet ou non. Si le fichier est complet, le drapeau est mis à vrai, sinon il est mis à

faux ;

- *Le bitfield.* Il s'agit du même champ que décrit précédemment et permet au nœud requête de savoir les morceaux disponibles de la source partielle.

Une fois le message FREP construit, il doit être envoyé au nœud qui a initié la requête correspondante. Pour s'y faire, le nœud envoie le message FREP au dernier saut du message FREQ correspondant. Lorsqu'un nœud reçoit un message FREP, il va d'abord vérifier s'il s'agit de la réponse d'une de ses requêtes. Si la réponse ne lui ai pas destinée, il examine la liste des fichiers. Pour chaque fichier contenu dans le message, si le nœud possède tous les morceaux indiqués, il va supprimer l'entrée de la liste des fichiers ans le message. Il est inutile que le nœud requête reçoive cette source puisque le nœud intermédiaire peut le desservir. Une fois le filtrage des fichiers terminé, il va vérifier dans sa table de routage pour l'entrée qui correspond à destination du message. Si l'entrée n'est pas trouvée, on considère que la route est brisée et un traitement de réparation doit est effectué (ce traitement est vu plus tard dans le chapitre). Si l'entrée est trouvée, le nœud cherche la paire « prochain saut, nombre de sauts » optimale, soit celle qui offre le plus petit nombre de sauts. Il va alors envoyer le message FREP à ce nœud. Si l'envoi n'est pas réussi (le nœud s'est déplacé par exemple), le nœud retire la paire de sa table de routage et réessaye avec la nouvelle meilleure paire. Si aucune paire ne s'y trouve, on considère que la route est brisée et un traitement de réparation doit être effectué. Lorsque le nœud destinataire reçoit la réponse, il peut alors afficher les résultats à l'usager Tout nœud recevant un message FREP met à jour sa table de transfert peu importe s'il est le destinataire du message. Si aucune entrée n'existe déjà pour le fichier, il en créé une. Il va ajouter le nœud source du message comme source pour le fichier.

TABLEAU 3.6 Message FREP

Champ	**Taille en octets**
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Numéro de séquence	4
Nombre de fichiers	1
Liste des fichiers	Variable

TABLEAU 3.7 Description d'un fichier dans un message FREP

Champ	Taille en octets
Taille du nom	4
Nom du fichier	Variable
Taille du fichier	6
Signature du fichier	16 (MD5)
État du fichier	1
Bitfield	Variable

3.6 Algorithme de transfert

La Figure 3.2 montre les étapes majeures de l'algorithme de transfert. Cet algorithme démarre lorsque l'usager choisit un fichier qu'il désire télécharger. Le protocole doit consulter la table de transfert pour trouver la meilleure source pour ce fichier. Les critères de comparaison pour les sources sont les suivantes, dans l'ordre :

1. Si le nombre de saut pour atteindre les sources est différent, on choisit la source avec le nombre de saut minimal ;
2. Si une des deux sources est une source complète, elle est choisie. Dans le cas où les deux sources sont complètes, le choix se fait aléatoirement ;
3. La source possédant le plus de morceaux est choisie.

La première étape d'un transfert est l'obtention des signatures de tous les morceaux du fichier. Pour s'y faire, le nœud envoie un message SREQ à la source. Le Tableau 3.8 montre la structure de ce message. L'unique champ propre à un message SREQ est donc l*a signature du fichier.*

L'envoi du message SREQ se fait de manière unicast. Ainsi, chaque nœud consulte sa table de routage afin de trouver le prochain saut vers la destination et lui envoie

TABLEAU 3.8 Message SREQ

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
Signature du fichier	16 (MD5)

le message. Lorsque le nœud destination reçoit le message SREQ, il doit le traiter et retourner un message SREP, qui est la réponse. Le Tableau 3.9 montre la structure d'un tel message. Un message SREP contient donc, en plus des champs génériques, le champ représentant la liste des signatures. Il s'agit d'une chaîne d'octets correspondant à la concaténation de toutes les signatures des morceaux, dans l'ordre.

L'envoi du message SREP se fait de la même manière que le message SREQ. Lorsqu'un nœud reçoit un SREP qui lui est destiné, il doit premièrement s'assurer de la validité des signatures des morceaux. Pour s'y faire, il applique la fonction de hachage à la chaîne reçue (la concaténation des signatures). Il devrait alors obtenir la signature du fichier. Si la valeur obtenue diffère de la signature du fichier, il y a problème. La source est retirée de la liste et le nœud doit alors envoyer un autre message SREQ à la prochaine meilleure source. Il peut également ajouter la source qui lui a retourné les mauvaises informations dans sa table noire, considérant ainsi la source comme malicieuse.

Une fois les signatures des morceaux obtenues, le nœud peut alors débuter son transfert. Le transfert s'effectue morceau par morceau. L'ordre du transfert des morceaux est aléatoire. Le nœud choisit donc un morceau aléatoirement parmi les morceaux qui lui manquent et les morceaux que la source possède. Si la source ne possède aucun morceau intéressant, elle est retirée et une nouvelle source est trouvée dans la table de transferts. Si aucune source n'existe, le nœud doit alors démarrer une recherche pour trouver de nouvelles sources (plus de détails sur le rafraichissement des sources sont donnés plus tard dans le chapitre). S'il existe une source avec au moins un morceau intéressant, le nœud choisit un morceau aléatoirement parmi les candidats et construit un message PREQ. Le Tableau 3.10 montre la structure d'un tel message. Un message PREQ contient donc, en plus des champs génériques, les champs suivants :

TABLEAU 3.9 Message SREP

Champ	**Taille en octets**
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
Liste des signatures	Variable * 16 (MD5)

- *La signature du fichier* ;
- *Le morceau requis.* Il s'agit d'un entier non signé indiquant le morceau que le nœud désire recevoir, le premier morceau correspondant à la valeur 0.

Le message PREQ est envoyé de la même manière que les autres messages unicast, soit en consultant la table de routage. Lorsqu'un nœud reçoit un message PREQ, il vérifie en premier lieu s'il possède le fichier et, le cas échéant, s'il possède le morceau requis. Si oui, le nœud peut alors traiter la requête et ce, même s'il n'est pas le destinataire. Ce scénario peut survenir si par exemple un nœud intermédiaire décide de télécharger le fichier après le nœud source. Si un nœud ne peut pas traiter la requête, soit à cause du fait qu'il ne possède pas le fichier ou bien qu'il ne possède pas le morceau requis, Il doit relayer le message PREQ au prochain saut. S'il n'est pas possible d'atteindre le prochain saut, on considère que la route est brisée et un traitement de réparation doit être effectué. Lorsqu'un nœud traite un PREQ, il doit retourner les données demandées au nœud source du message. Pour s'y faire, il construit un message PREP. Il existe deux types de messages PREP, soit le message PREPC et PREPI. Si la source est une source complète, elle construit un message PREPC. S'il s'agit d'une source incomplète, elle construit un message PREPI. Si la taille du morceau (incluant l'entête du message) est supérieure à la taille d'un paquet de transfert, plusieurs messages PREP doivent être envoyés successivement. Chaque message va contenir un bloc du morceau. Le Tableau 3.11 montre la structure d'un tel message. Un message PREPC contient donc, en plus des champs génériques, les champs suivants :

- *La signature du fichier* ;
- *Le morceau requis.* Ce champ doit contenir la même valeur que dans le message

TABLEAU 3.10 Message PREQ

Champ	**Taille en octets**
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Signature du fichier	16 (MD5)
Morceau requis	4

PREQ correspondant ;

- *L'offset du bloc.* Il s'agit d'un entier non signé indiquant la position dans le morceau où le bloc débute ;
- *La taille du bloc.* Il s'agit d'un entier non signé indiquant la taille du bloc. Dans le cas où le message contient tout le morceau, ce champ sera égal à la taille du morceau ;
- *Les données.* Il s'agit d'une chaîne d'octets représentant les données du fichier.

Le Tableau 3.12 montre la structure d'mn message PREPI, qui ne se différencie d'un message PREPC que par l'ajout d'un champ additionnel, soit le bitfield. Ce bitfield correspond à celui de la source incomplète pour le fichier transféré.

Le champ additionnel envoyé par une source incomplète permet au client de mettre à jour le bitfield de la source dans sa table de transfert. Ainsi, si la source incomplète télécharge le fichier en parallèle, le client est mis au courant de l'avancement du transfert.

Une fois le message PREP construit, il est envoyé de manière unicast vers la source du message FREP correspondant. Lorsque le client reçoit le FREP qui correspond à son FREQ, il doit le traiter. Ainsi, il va écrire sur le disque le morceau ou le bloc du morceau reçu. S'il s'agit d'un morceau entier ou bien s'il s'agit du dernier bloc d'un morceau, le nœud calcule la signature du morceau et s'assure que la signature correspond à celle qu'il avait reçue. Si les signatures diffèrent, il y a eu un problème de transmission qui peut être dû à un problème physique ou encore à un nœud malicieux. Il retire alors la source de sa liste et redemande le morceau à la prochaine meilleure source. Si les signatures sont identiques, le transfert de ce morceau est considéré réussi et le nœud identifie ce morceau comme étant disponible dans sa table de transfert, en mettant le drapeau correspondant à ce morceau à vrai dans le bitfield et en incrémentant le nombre de paquets possédés. Il peut désormais desservir ce bloc aux autres clients qui le désirent. Le nœud va également s'assurer de mettre à jour dans sa table de transfert les informations relatives à la source. Il doit en premier lieu vérifier si le nœud source du message FREP fait partie de sa liste de sources pour le fichier. Si tel n'est pas le cas, le nœud qui lui a envoyé le message n'est pas la source vers laquelle était destinée la requête, mais plutôt un nœud intermédiaire. Dans ce cas-ci, le nœud est ajouté à la liste des sources potentielles. Comme il s'agit d'un nœud intermédiaire, il va devenir la nouvelle source la plus proche, puisqu'il est forcément plus proche. Si le message FREP est un message FREPC, la source est

TABLEAU 3.11 Message PREPC

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Signature du fichier	16 (MD5)
Morceau requis	4
Offset du bloc	4
Taille du bloc	4
Données	Variable

TABLEAU 3.12 Message PREPI

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Signature du fichier	16 (MD5)
Morceau requis	4
Offset du bloc	4
Taille du bloc	4
Données	Variable
Bitfield	Variable

considérée comme source complète et peut desservir tout le fichier. Si le message est un message FREPI, la source est une source incomplète et peut desservir uniquement les morceaux dont leur drapeau est mis à vrai dans le bitfield contenu dans le message FREPI. Dans ce cas-ci, le nœud s'assure de mettre à jour cette information dans la table de transfert.

Lorsqu'un nœud termine de recevoir un morceau (soit en recevant un message FREP qui contient un morceau complet ou bien en recevant tous les messages FREP qui contiennent les blocs du morceau), il demande alors le prochain morceau. Si aucun morceau n'est requis, donc le fichier est complet, le transfert est terminé et le fichier est marqué comme entier dans la base de données. Il peut donc désormais répondre aux autres nœuds comme étant une source complète pour ce fichier.

Un nœud intermédiaire peut également garder une copie d'un message PREP qui ne lui est pas destiné, si ce message est intéressant. Il peut alors le traiter comme s'il s'agit de son message, en écrivant sur le disque les données contenues dans ce message. Il peut alors également mettre à jour la liste des sources pour ce transfert.

3.7 Rafraichissement

Afin de s'assurer d'avoir une liste de sources à jour dans sa table de transfert, tout client peut, pendant un transfert, diffuser un paquet de rafraichissement afin de vérifier si une source, qui n'était pas disponible au début du transfert, le devient. Pour s'y faire, un message FREF est envoyé périodiquement. Le Tableau 3.13 montre la structure d'un tel message. Le message FREF est donc semblable au message FREQS, avec comme unique différence l'ajout d'un champ supplémentaire, le bitfield. En précisant les morceaux qu'il possède déjà, le nœud évite de recevoir les réponses de sources partielles inutiles (qui ne possèdent donc pas de morceaux intéressants).

Pour limiter la diffusion, le champ TTL doit être mis à la valeur correspondante au nombre de sauts requis pour atteindre la meilleure source actuelle, moins un. Ainsi, si le nœud reçoit des réponses, elles ne proviendront que de sources plus proches. Si ces sources sont complètes ou possèdent au moins un morceau intéressant, elles deviendront les nouvelles candidates pour la meilleure source, pour le prochain morceau demandé. De plus, le nœud doit s'assurer d'envoyer un message FREF avec un numéro de séquence unique, sinon le message sera ignoré par ses voisins.

Lorsqu'un nœud reçoit un message FREF, il le traite comme un message FREQS,

TABLEAU 3.13 Message FREF

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Numéro de séquence	4
Nombre de sauts	1
Signature	6 (MD5)
Bitfield	Variable

avec comme seule différence la vérification du bitfield. Ainsi, le nœud ne retourne une réponse que s'il possède au moins un morceau intéressant au client. La réponse retournée est un message FREP (FREPI ou FREPC), de la même manière que pour une réponse à un FREQ.

3.8 Reprise d'un transfert

Un transfert peut être interrompu intentionnellement par un usager ou encore accidentellement lors d'une panne par exemple. Il est donc intéressant d'offrir à l'usager la possibilité de reprendre son transfert sans devoir tout télécharger de nouveau. Ainsi, le bitfield qui indique les blocs déjà possédés doit être mémorisé sur une mémoire non volatile. À chaque fois qu'un nouveau bloc est reçu, en plus de mettre à jour le bitfield en mémoire, il faut le mettre à jour sur la mémoire non volatile également. Ainsi, si le transfert est arrêté et l'application fermée, cette information n'est pas disparue. Lors de la reprise du transfert, l'application peut alors lire cette information et en construire le bitfield. L'information sur les blocs peut être stockée avec les signatures, s'il s'agit de l'implémentation choisie pour la disponibilité des signatures.

En plus de sauvegarder le bitfield, le nœud peut également sauvegarder sa table de transfert contenant notamment la liste des sources disponibles. Cette technique n'est pas nécessaire et n'est pas nécessairement recommandée, puisque les liens risquent de changer rapidement si la mobilité du réseau est forte. Il est donc préférable de reconstruire la liste de sources disponibles en envoyant une requête. La requête utilisée est un message FREF, en utilisant une technique progressive comme avec les messages

FREQ.

Cette fonctionnalité n'est pas vitale au protocole. Si elle n'est pas implémentée, le protocole ne pourra reprendre les transferts. Ainsi, les transferts non terminés doivent être recommencés du départ.

3.9 Maintenance des routes

Dans le protocole proposé, les paquets sont envoyés selon le meilleur chemin possible pour atteindre la destination. Ainsi, la table de routage est consultée afin de trouver le meilleur chemin possible vers une destination. Si plusieurs chemins existent, celui avec la meilleure métrique, soit le nombre de sauts minimal, est choisi. Le paquet est donc envoyé au prochain saut de ce chemin. Si l'envoi n'est pas réussi à cause d'un bri de lien, le paquet doit être renvoyé. Le nœud retire alors le meilleur prochain saut de sa table, puisqu'il n'est plus valide. Il va ensuite trouver le nouveau meilleur prochain saut disponible et lui envoyer le paquet. La réparation des liens brisés effectuée est donc un traitement local, contrairement aux protocoles de routages réactifs tel qu'AODV, qui effectuent des réparations globales. Les auteurs d'ORION ont démontré qu'une réparation locale offre de meilleures performances qu'une réparation globale. Si le nœud intermédiaire qui doit réparer une route brisée ne possède pas de routes alternatives, le traitement local n'est pas possible. Il doit alors prendre action dépendamment du type du message. Pour les réponses aux requêtes de fichiers, le nœud peut tout simplement ignorer l'erreur et se débarrasser du paquet. Ce scénario est, en pratique, rare car le processus de requête/réponse de fichiers est assez rapide et à moins que les nœuds ne soient en mouvement avec une grande vitesse, le scénario est peu probable. Pour une requête de données, le nœud doit avertir le nœud qui l'a envoyée du bris afin que ce dernier puisse continuer son transfert à partir d'une source alterne. Pour s'y faire, un message ROUTEER est envoyé. Le Tableau 3.14 montre la structure d'un tel message. Un message ROUTEER contient donc comme unique champ additionnel la signature du fichier. Il s'agit d'une chaîne d'octets correspondant à la signature du fichier recherché. La taille de ce champ est fixe et dépend de la fonction de hachage utilisée (16 octets pour MD5).

Lorsqu'un nœud reçoit un message ROUTEER qui lui est destiné, il va être averti que la source actuelle n'est plus valide et doit la retirer de sa liste de source. Ensuite, il doit exécuter de nouveau le processus de demande de morceau. Si un nœud

TABLEAU 3.14 Message ROUTEER

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Signature du fichier	16 (MD5)

intermédiaire ne peut réparer une route pour un message de type réponse de données, il peut alors ignorer le message.

Afin de minimiser les délais, il est impératif que le protocole soit averti d'un bri le plus tôt possible. L'implémentation du protocole peut profiter de la signalisation au niveau de la couche liaison, si cette dernière l'offre. Tel est le cas pour le protocole 802.11. Si la signalisation n'est pas présente ou ne peut pas être implémentée, on peut envisager d'accuser la réception des messages au niveau application. Ainsi, pour chaque message envoyé, le nœud récipiendaire envoie un message ACK pour signaler au nœud envoyeur qu'il a bel et bien reçu le message. Le Tableau 3.15 montre la structure d'un tel message. Le seul champ propre à un message ACK est donc le type du message accusé. Il s'agit d'un octet non signé qui correspond au type du message que le nœud accuse.

Il est à noter que les champs Source et Destination doivent être égaux à ceux du message accusé et que les champs Dernier Saut et TTL ne sont pas utilisés et peuvent être retirés de l'implémentation si la performance est primordiale. Si le nœud ne reçoit pas l'accusé dans un certain délai, il considère que le lien est brisé et débute

TABLEAU 3.15 Message ACK

Champ	Taille en octets
Type	1
Destination	4 (IPv4)
Source	4 (IPv4)
Dernier saut	4 (IPv4)
TTL	1
Type du message accusé	1

la réparation de la route. Il est également à noter que les messages FREP et FREF ne peuvent être accusés, tant au niveau matériel qu'au niveau application car il s'agit de messages diffusés sans récipiendaires précis.

3.10 Pertes de paquets

Même avec le mécanisme de réparation de routes, il peut y arriver qu'un client ne reçoive pas la réponse à sa requête lors d'un transfert. Peu importe la raison, avec le protocole décrit comme tel, le nœud se retrouve bloqué en attente de la réponse. Il faut donc prévoir un mécanisme pour régler ce problème. Pour s'y faire, le nœud accord un certain délai pour chaque requête. S'il ne reçoit pas de réponse dans ce délai, il considère le paquet perdu et doit reprendre son transfert. Il est à noter qu'une réponse à une requête peut être un message ROUTEER indique au nœud que la route est brisée. Lorsqu'il y a une perte de paquet, le nœud doit considérer la source comme n'étant plus disponible et la retirer de la liste. Ensuite, il peut exécuter de nouveau la requête de morceau.

Il est à noter qu'une perte de paquet au niveau d'une requête de fichier n'est pas grave. Dans le pire scénario, le nœud ne reçoit pas la réponse d'une source intéressante. Avec le mécanisme de rafraîchissement des sources, discuté plus haut, ce scénario devient peu problématique.

3.11 Coopération inter-couches

Un des objectifs du protocole proposé est sa neutralité vis-à-vis des autres protocoles opérant sur le mobile (notamment le protocole de routage), donc la possibilité de le déployer sans apporter de modification à d'autres couches et sans être forcé à utiliser un protocole en particulier. En effet. La solution peut opérer parfaitement à partir de la couche 7, en encapsulant tous ses messages dans des paquets UDP. Ainsi, on n'est nullement limité par le protocole de routage à utiliser. Cependant, dans un environnement ad hoc, la performance est souvent plus importante qu'une bonne architecture en couche. Ainsi, il n'est pas rare de voir les systèmes opérer en mode inter couches, déviant ainsi du standard OSI utilisé presque exclusivement dans d'autres réseaux. Dans une architecture inter couches, la communication entre les couches n'est plus exclusivement réservées entre les couches adjacentes ; ainsi, la couche 7

peut par exemple communiquer avec la couche 3 et ce, de manière directe sans passer par les couches intermédiaires. Cette technique évite des échanges inutiles entre les couches intermédiaires. De plus, dans un réseau ad hoc, tout mobile joue également le rôle de routeur. L'architecture avec communication inter couches est encore plus intéressante dans un tel réseau puisqu'elle permet d'éliminer des calculs redondants, ce qui augmente la performance et réduit donc la consommation d'énergie.

Le protocole proposé offre plusieurs possibilités d'implémenter des communications inter couches. Le protocole implémente sa propre table de routage qu'il utilise pour acheminer ses messages. De plus, ces messages contiennent toutes les informations nécessaires pour l'acheminement (par exemple les champs TTL). Pour cette raison, il est inutile d'encapsuler les messages dans des paquets UDP. Le protocole peut donc communiquer directement avec la couche 3 (le protocole IP) pour envoyer ses paquets, évitant ainsi toutes les couches intermédiaires. On réduit donc le temps de traitement des paquets (inutile d'encapsuler, de décapsuler, de relayer le message de couche en couche) et le débit utilisé (plus besoin d'un entête UDP). Il faut alors modifier la couche 3 afin que le protocole IP reconnaisse les messages destinés au protocole et puisse les lui transmettre et ce, de manière directe.

Comme le protocole achemine lui-même ses messages, tous les messages envoyés sont des diffusions ou des envois unicast à des voisins. Ainsi, il est complètement inutile d'effectuer une recherche de route dans le cas d'un protocole réactif ou encore de consulter la table de routage dans le cas d'un protocole réactif. Il est donc intéressant de modifier la couche 3 afin qu'elle envoie les messages provenant de notre protocole directement, sans faire appel au protocole de routage. De plus, lors d'une panne ou d'une collision, la couche 3 peut avertir directement notre protocole que le paquet n'a pu être envoyé. Dans le cas où le protocole de routage (proactif ou réactif) maintient une table de routage, il peut être envisageable de supprimer la table de routage de notre protocole et d'utiliser celle du protocole de routage. Ainsi, lorsque des requêtes ou réponses sont traitées, le protocole P2P informe le protocole de routage des routes obtenues. Cette implémentation est cependant complexe et n'est profitable que pour quelques protocoles de routages (notamment AODV).

3.12 Transferts simultanés

Un client peut, s'il le désire, démarrer 2 ou plus de transferts simultanément. Dans un environnement sans fil, il ne serait pas judicieux d'envoyer des requêtes de morceaux simultanément sinon il y a risque de congestion au niveau radio et donc de collisions. Ainsi, l'application va, lors de transferts simultanés, demander les morceaux de tous les fichiers de manière séquentielle. Lorsqu'un morceau d'un fichier termine, l'application demande un morceau pour le prochain transfert et ainsi de suite. On suit donc une répartition par permutation circulaire. Si l'application envoie une requête pour chaque fichier par ronde, on parle alors d'une répartition équilibrée où chaque fichier reçoit une attention égale. L'usager peut décider d'augmenter la priorité d'un fichier par rapport aux autres ou encore la diminuer. L'application peut par exemple offrir les cinq priorités suivantes pour chaque transfert :

- *Très haute.* Avec cette priorité, un fichier se voit attribuer trois demandes de morceaux par ronde.
- *Haute.* Avec cette priorité, un fichier se voit attribuer deux demandes de morceaux par ronde.
- *Normale.* Avec cette priorité, un fichier se voit attribuer une demande de morceaux par ronde. Il s'agit de la priorité par défaut des transferts qui démarrent.
- *Basse.* Avec cette priorité, un fichier se voit attribuer une demande de morceaux par deux rondes.
- *Très basse.* Avec cette priorité, un fichier se voit attribuer une demande de morceaux par trois rondes.

3.13 Scénario de recherche de fichier

Pour illustrer l'algorithme de recherche, considérons le scénario mobile présenté à la Figure 3.3. Il est composé de quatre nœuds mobiles possédant chacun une base de fichiers parmi 4 fichiers possibles. On suppose que toutes les sources sont complètes.

Supposons que le mobile 1 lance une recherche correspondant aux fichiers 2 et 3. Le message FREP est diffusé par le mobile 1 à tous ses voisins, soit au mobile 3, tel qu'indiqué dans la Figure 3.4. Comme il s'agit d'une nouvelle requête, le mobile 3 la traite. Il commence par rediffuser le message à ses voisins, soient les mobiles 1, 2 et 4 (Figure 3.5). Ensuite, il met à jour sa table de routage est consulte sa table de

fichiers. Comme il possède le fichier 2, il doit retourner une réponse, soit un message FREP. En consultant sa table de routage, le prochain saut pour atteindre le mobile 1 est le mobile 1 lui-même, et l'envoi s'effectue de cette manière, comme montré à la Figure 3.6. De la même manière, les mobiles 2 et 4 traitent le message FREQ. Lorsque le mobile 3 reçoit la diffusion du FREQ des mobiles 2 et 4, il ignore ces requêtes puisqu'elles ne sont pas nouvelles. Le mobile 4 ne possède pas de fichiers qui satisfont les critères de la recherche. Le mobile 2 quant à lui, possède les fichiers 2 et 3 et doit donc retourner une réponse. Après la construction du message FREP, il consulte sa table de routage et trouve le prochain saut vers le mobile 1, soit le mobile 3. Il envoie donc le message FREP au mobile 3, tel qu'illustré à la Figure 3.7. Le mobile 3 qui reçoit le message va le filtrer en retirant le fichier 2, puisqu'il le possède. Il va ensuite consulter sa table de routage pour trouver le prochain saut vers le mobile, soit le mobile 1 lui-même, et relayer le message (Figure 3.8).

La Figure 3.9 montre les différents messages envoyés lors de la recherche effectuée. Une fois la recherche terminée, l'usager du mobile 1 est présenté avec deux fichiers potentiels, soit le fichier 2 avec comme sources les mobiles 2 et 3, et le fichier 3 avec comme source le mobile 2. C'est alors que l'usager peut effectuer son choix. Supposons qu'il choisisse le fichier 3. Il va alors consulter sa table de transfert pour trouver la meilleure source pour ce fichier. Comme il n'existe qu'une seule source, soit le mobile 2, elle sera automatiquement choisie. Si le fichier demandé était le fichier 2, il aurait le choix entre les mobiles 2 et 3, le dernier étant la meilleure source puisque la route est plus courte. Une fois la source déterminée, la première étape est de demander les signatures des morceaux. Pour s'y faire, le mobile 1 construit le message SREQ et l'envoie au mobile 2, en passant par le mobile 3, qui est le prochain saut trouvé dans la table de routage (Figure 3.10).

Le mobile 2 va alors construire la réponse et l'envoyer au mobile 1, de la même manière (Figure 3.11). Une fois les signatures reçues et validées, le mobile 1 peut alors envoyer les requêtes de morceaux et le mobile 2 va renvoyer les réponses correspondantes, de la même manière que pour la requête des signatures du fichier. Une fois le transfert terminé, l'usager peut alors utiliser le fichier. La Figure 3.12 montre les paquets échangés lors du transfert.

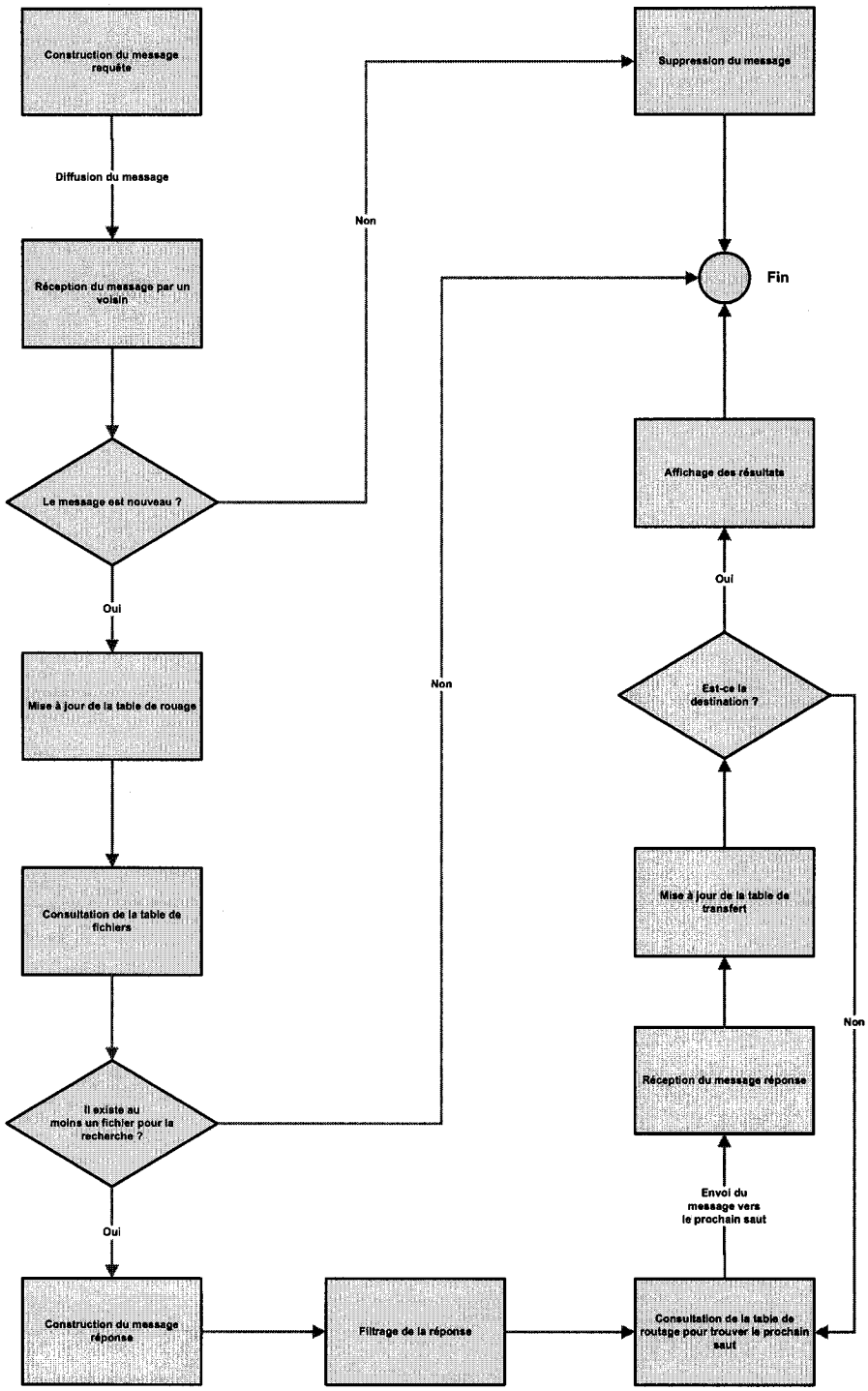


FIGURE 3.1 Algorithme de recherche de fichier

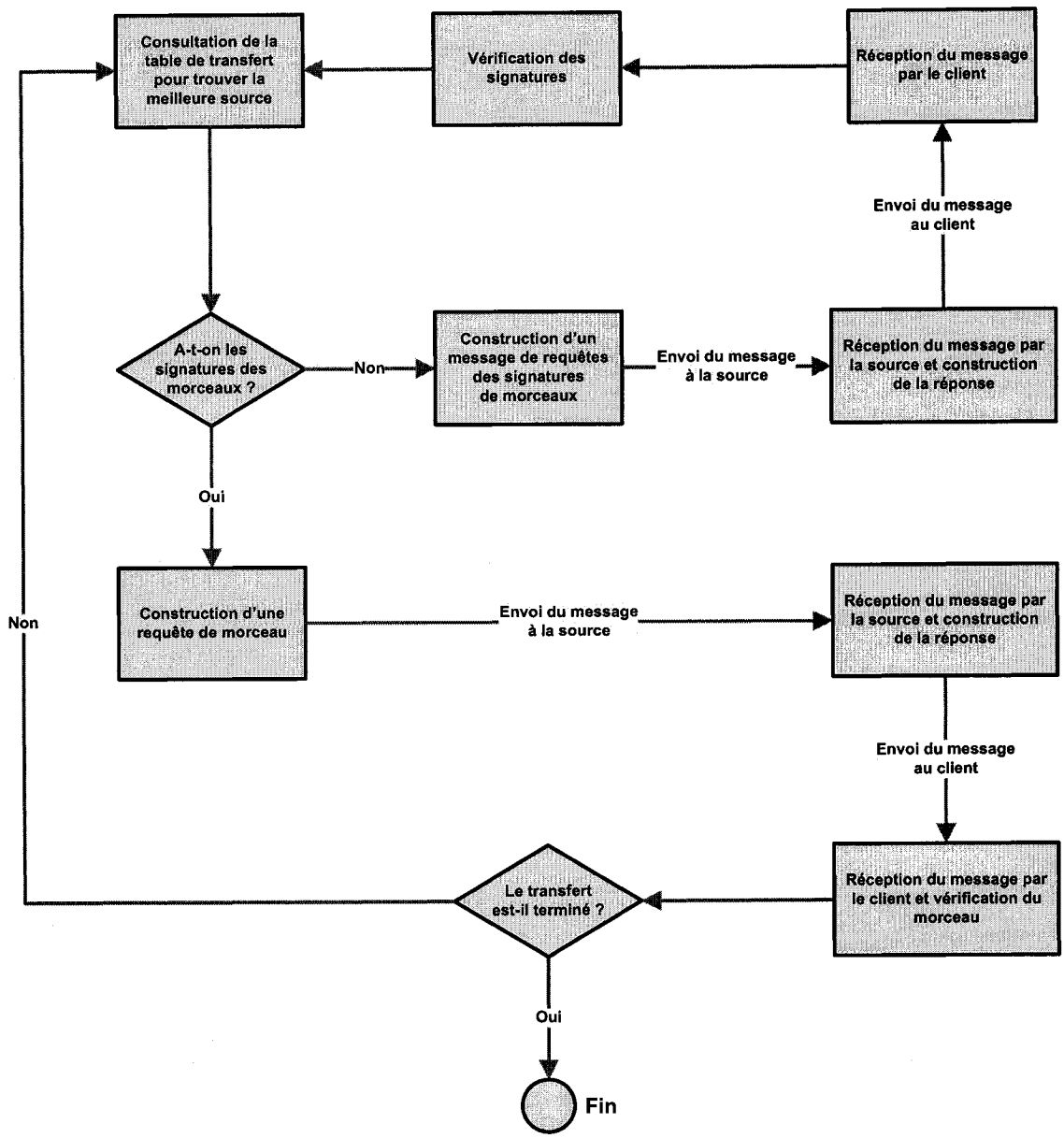


FIGURE 3.2 Algorithme de transfert

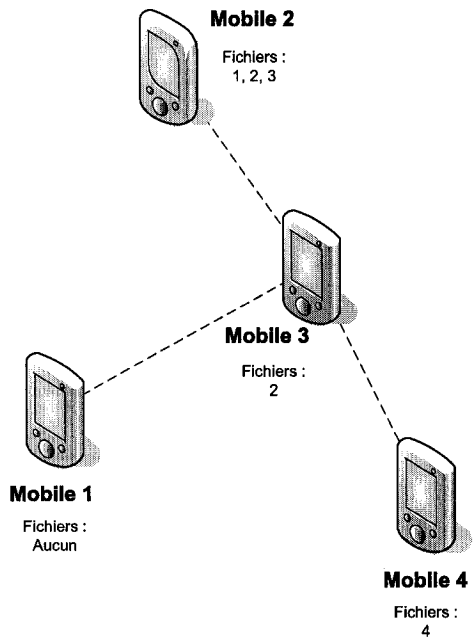


FIGURE 3.3 Réseau mobile ad hoc

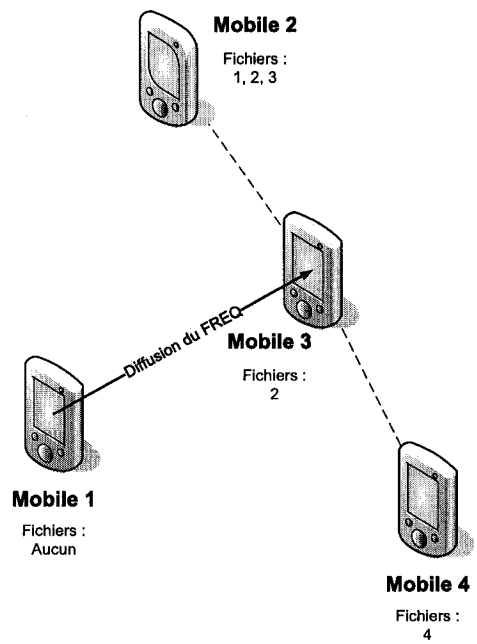


FIGURE 3.4 Diffusion du FREQ par le mobile 1

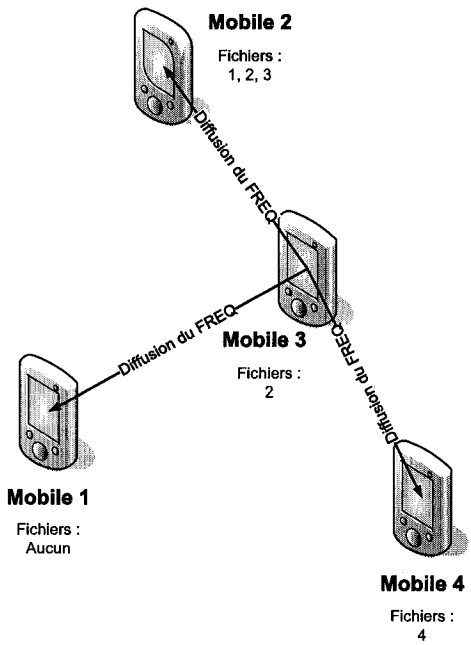


FIGURE 3.5 Diffusion du FREQ par le mobile 3

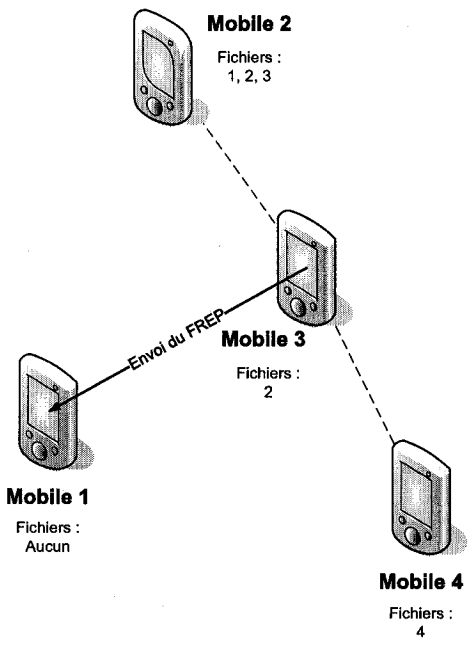


FIGURE 3.6 Envoi du FREP par le mobile 3

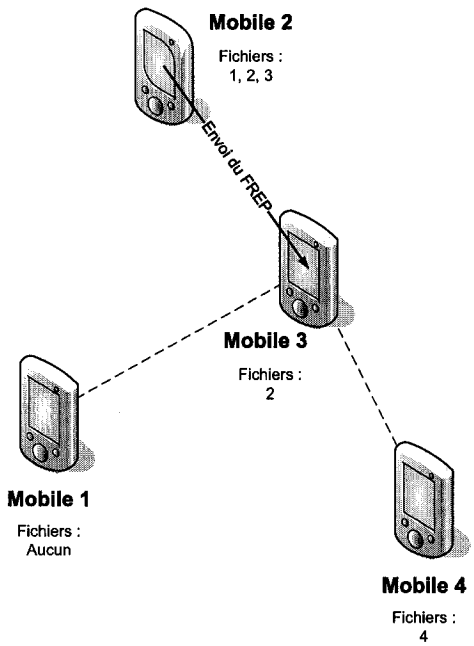


FIGURE 3.7 FREP du mobile 2 au prochain saut

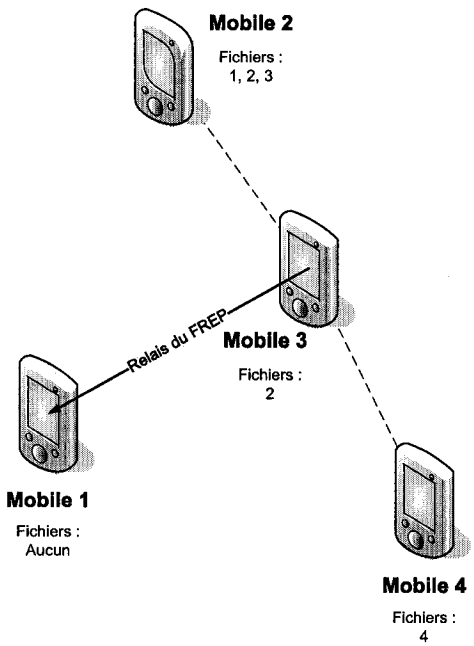


FIGURE 3.8 Relais du FREP du mobile 3 vers le mobile 2

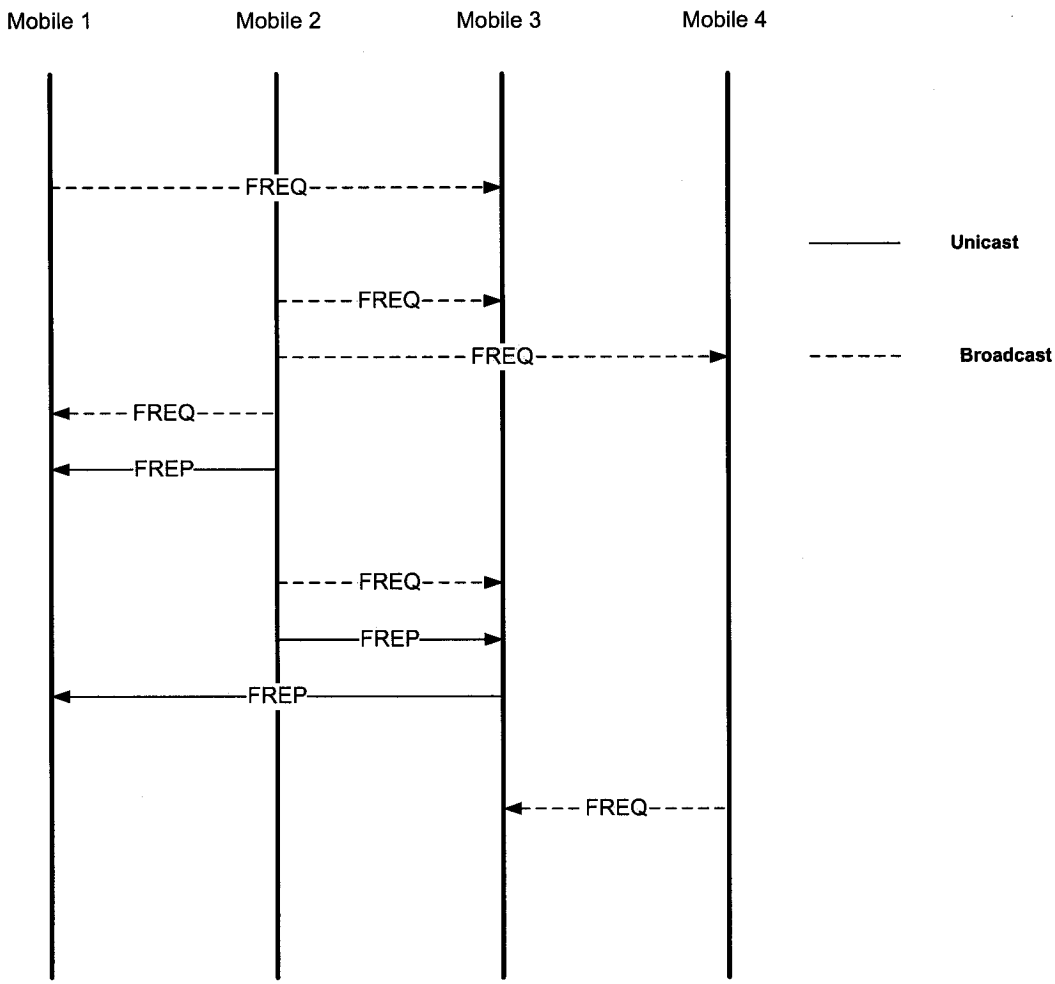


FIGURE 3.9 Messages transmis lors de la recherche

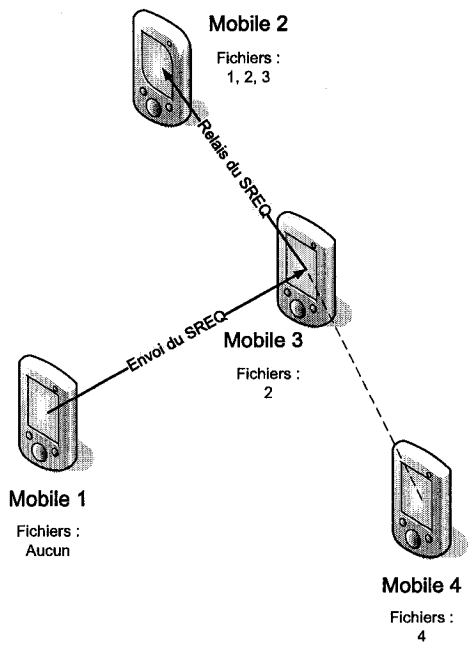


FIGURE 3.10 Envoi du SREQ au mobile 2

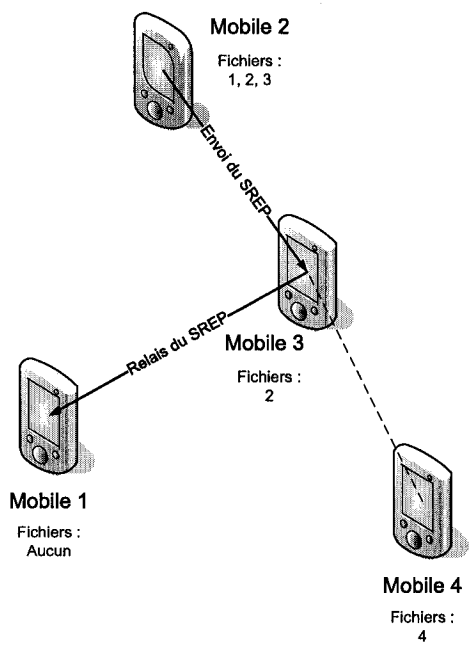


FIGURE 3.11 Envoi du SREP au mobile 1

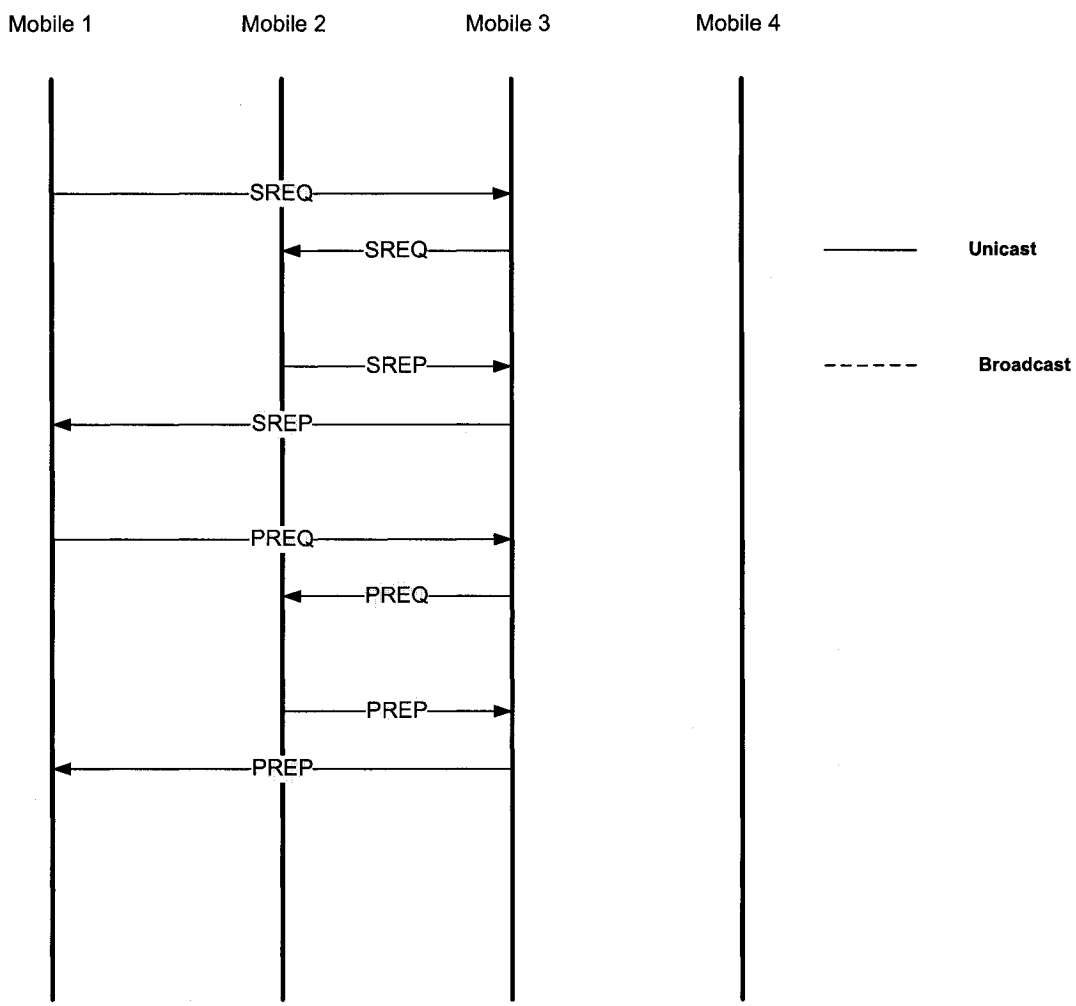


FIGURE 3.12 Paquets envoyés lors du transfert

CHAPITRE 4

ANALYSE DE PERFORMANCE

Dans le chapitre précédent, nous avons proposé un protocole de partage de fichiers pair à pair. Dans ce chapitre, nous allons procéder à l'évaluation de performance de la solution proposée que nous comparons avec la performance du protocole ORION. Plus précisément, nous allons évaluer la performance lors des transferts de fichiers. Cette évaluation sera réalisée en deux parties. Premièrement, un modèle analytique sera développé et une évaluation des résultats obtenus avec ce modèle sera effectuée. Ensuite, un modèle expérimental sera présenté. Nous effectuerons donc l'implémentation et la simulation des protocoles dans un logiciel de simulation de réseaux ad hoc et une évaluation des résultats de ces simulations sera effectuée.

4.1 Modèle analytique

Dans cette partie, un modèle mathématique sera développé afin de pouvoir évaluer la performance des deux protocoles. Ainsi, à l'aide d'expressions mathématiques, nous pourrons déduire les performances théoriques, anticipant ainsi les résultats expérimentaux.

4.1.1 Hypothèses

Afin de simplifier le modèle analytique, nous avons émis quelques hypothèses. Le modèle analytique considère qu'un réseau est composé de m mobiles, chacun déployant le protocole en question. Parmi ces mobiles, on retrouve une source complète (le nœud 1) pour un fichier donné et m-1 clients potentiels. Seule la performance de l'algorithme de transfert sera étudiée. On étudiera notamment la performance au niveau du temps de téléchargement du fichier. On suppose que le temps requis par un paquet pour effectuer un saut est constant, peu importe la distance physique et sa taille. De plus, le temps de traitement d'une requête de morceau par une source est supposée constante (en réalité, elle dépend de la charge de la source, donc du nombre de client qu'elle

dessert). On suppose également une transmissions parfaite, sans erreurs ni collisions, tout au long des transferts.

4.1.2 Constantes et variables du modèle analytique

Les variables et constantes qui constitueront les expressions mathématiques de notre modèle sont, pour un client donné :

- N, le nombre de morceaux qui composent le fichier ;
- *Hops(x, y)*, la fonction qui retourne le nombre de sauts requis pour atteindre la source s ;
- *Trait*, le temps de traitement d'une requête ;
- μ, le temps requis à un paquet pour traverser un saut.

4.1.3 Expressions mathématiques

ORION

Pour le protocole ORION, le temps de téléchargement d'un fichier par une source dépend uniquement des sources complètes. En effet, le nœud va demander à la meilleure source complète tous les morceaux. Dans notre cas, il n'existe qu'une seule source complète, soit le nœud 1. Pour chaque morceau, une requête est envoyée, suivi de la réponse. L'expression mathématique de la performance d'un transfert avec le protocole ORION est donc, pour un nœud x :

$$Temps = 2 * N * \mu * Hops(1, x) + N * Trait$$

Comme μ et *Trait* sont des constantes, la performance d'ORION ne dépend que du nombre de sauts pour atteindre la source et du nombre de morceaux qui composent le fichier.

Solution proposée

Pour la solution proposée, le temps de téléchargement dépend des sources complètes et des sources partielles. En effet, le nœud va demander à la meilleure source, complète ou non, un morceau tant qu'elle en possède au moins un intéressant. De plus, il faut considérer l'étape supplémentaire, soit l'échange des signatures de morceaux.

Cette étape sera modélisée en incrémentant le nombre de morceaux. L'expression mathématique de la performance d'un transfert avec le protocole ORION est donc, pour un nœud x :

$$Temps = \sum_{n=1}^{N+1}(2 * \mu * Hops(y,x) + Trait)$$

avec y la meilleure source intéressante au moment de la requête d'un morceau.

4.1.4 Plan d'expériences et de simulation

Cette section présente le plan d'expériences qui sera utilisé pour évaluer les performances des deux protocoles suivant le modèle analytique. Nous commencerons par énoncer la configuration de la simulation. Ensuite, nous présenterons l'analyse des résultats.

Configuration de la simulation

Nous avons fixé la valeur N à 200 morceaux, la valeur Trait à 100ms et la valeur μ à 50ms. Nous avons fait varier la fonction *Hops*(1,x)de 1 à 5, avec un incrément de 1. De plus, pour la solution proposée, nous avons varié la probabilité, P, que la meilleure source soit partielle pour un morceau de 0% à 25% et à 50%. Ainsi, pour chaque requête, on retrouve avec un une probabilité P une source partielle y plus proche que la source complète (le nœud 1). On considère alors que *Hops*(y, x) est une valeur aléatoire comprise dans l'intervalle [1 ; *Hops*(1,x)]. Le Tableau 4.1 résume la configuration des principaux paramètres utilisés durant les simulations. Il est à noter que les paramètres dynamiques, qui changent d'une simulation à l'autre, sont surlignés en gris.

TABLEAU 4.1 Configuration des paramètres de simulation du modèle analytique

Paramètre	Valeur(s)
N	20
Trait	100ms
μ	50ms
Hops(1,x)	1, 2, 3, 4, 5
P	0%, 25%, 50%

Les simulations ont été effectuées à l'aide d'une implémentation des équations en C++. Puisque l'équation de la solution proposée contient des valeurs aléatoires, plusieurs itérations ont été effectuées et les valeurs utilisées sont les moyennes de résultats de ces itérations.

4.1.5 Analyse des résultats de simulation

La Figure 4.1 montre l'évolution du temps de transfert lorsqu'on augmente le nombre de sauts qui séparent le client de la source.

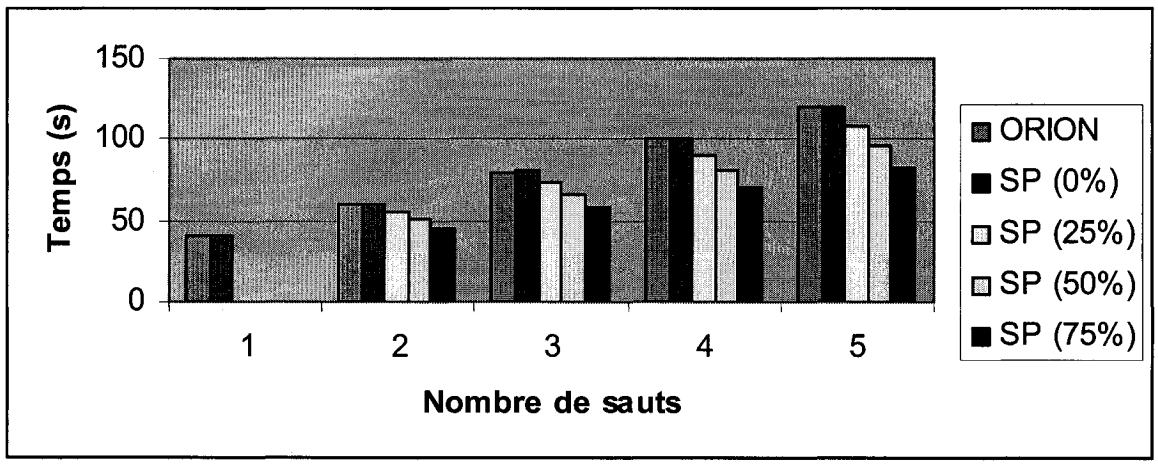


FIGURE 4.1 Temps de transfert en fonction du nombre de sauts

On remarque premièrement que le protocole ORION et la solution proposée avec un P de 0% se comportent de manière relativement identique. En effet, lorsqu'on fixe P à 0%, on suppose que tout au long du transfert, la meilleure source est la source complète. Ainsi, l'expression de la performance de la solution proposée devient :

$$Temps = \sum_{n=1}^{N+1}(2 * \mu * Hops(1,x) + Trait) = 2 * (N+1) * \mu * Hops(1,x) + N * Trait$$

On remarque donc que la seule différence avec ORION est alors l'échange des signatures, ce qui augmente le nombre de morceaux de 1. Cependant, cette étape est considérée négligeable par rapport au transfert complet, surtout lorsque N est grand. Ainsi, dans ce cas, qui est le pire cas pour la solution proposée, les protocoles se comportent de manière identique.

Plus la probabilité P augmente, meilleure devient la performance de la solution proposée. En effet, si à un moment donné lors du transfert y n'est pas le nœud 1, *Hops*(y,x) est alors inférieur à *Hops*(1,x). Ainsi, la performance de la solution proposée devient automatiquement meilleure que celle d'ORION. Plus ce phénomène se produit lors du transfert, plus la performance du protocole est améliorée. Le modèle analytique nous permet donc de conclure que la solution proposée est au moins aussi bonne qu'ORION (dans le pire cas) et que dans un cas général, la performance devrait être supérieure. La performance de la solution proposée dépend donc de la disponibilité de meilleures sources partielles que de la meilleure source complète. Il faut également noter que le modèle mathématique est un modèle simplifié. En effet, avec ORION, toutes les requêtes sont envoyées au nœud 1, ce qui augmente le temps de traitement de ce nœud. Avec la solution proposée, les requêtes peuvent être envoyées à des sources partielles, réduisant ainsi la congestion au nœud 1 et augmentant davantage la performance de notre solution. Le temps de traitement d'un nœud n'est donc certainement pas une constante. La performance de la solution devrait être encore meilleure si on tient compte de ce phénomène.

4.2 Modèle expérimental

4.2.1 Environnement d'implémentation et de simulation

L'environnement de développement

L'implémentation et la simulation de la solution seront réalisées sur une plate-forme de type PC. La machine en question possède un processeur *Intel Pentium IV* de 2.40 Gigahertz (GHz) et une mémoire vive de 512 Mégaoctets (Mo). Le système d'exploitation sur la machine est la version *Windows XP Professional*, développée par *Microsoft*, à laquelle a été ajouté l'ensemble de modifications provisoires 2 (*service pack* 2, en anglais). La programmation et la compilation du code source se font à partir de l'éditeur *Visual C++ .NET 2003* et de l'engin *nmake*, respectivement, faisant partie de la suite *Visual Studio .NET (2003)* de *Microsoft*.

QualNet : un logiciel de simulation pour les réseaux mobiles ad hoc

Le logiciel QualNet est un simulateur spécialement conçu pour l'implantation et la simulation des réseaux sans fil intégrant une composante de mobilité. Ce simulateur est en fait la version commerciale du logiciel *GloMoSim* (*Global Mobile system Simulator*) développé par la *University of California at Los Angeles* (*UCLA*). Comme sa contrepartie universitaire, QualNet se base sur *PARSEC* (*PARallel Simulation Environment for Complex system*), un langage de simulation parallèle à événements discrets, similaire au C. Chaque protocole est conçu sous la forme d'une machine à états finis où chaque évènement généré modifie l'état du protocole, et par conséquent, celui du système.

Le simulateur possède une structure modulaire qui facilite les extensions et les modifications au code source. En effet, l'architecture du simulateur QualNet adopte une approche à base de couches. Les cinq couches du simulateur sont : applications, transport, réseau, liaison (MAC) et physique. La communication entre les protocoles de chaque couche se fait à partir d'évènements générés discrètement à différents moments de la simulation.

La couche application est responsable de générer les trafics. Elle implémente les protocoles FTP, TELNET et HTTP. Au même niveau, l'on retrouve également un générateur de trafic à taux constant ou *Constant Bit Rate* (CBR), en anglais. La couche Transport offre un service de transmission de bout-en-bout aux protocoles de la couche application. Les protocoles TCP, UDP et RSVP-TE peuvent être utilisés pour desservir les besoins de la couche application. La principale couche de l'architecture est la couche réseau qui se divise en deux composantes. La première est responsable de diriger les données vers leur destination, autrement dit de router les paquets, et inclut des protocoles réseau issus de l'Internet tel IP (*Internet Protocol*), OSPF (*Open Shortest Path First*), MPLS (*Multiple Path Label Switching*) aussi bien que des protocoles spécifiquement conçus pour les réseaux mobiles ad hoc comme AODV (*Ad hoc On-Demand Vector*) ou DSR (*Dynamic Source Routing*). La deuxième composante s'occupe de l'ordonnancement de la mise en file, queuing en anglais, sur chaque nœud simulé et propose des algorithmes tels FIFO (*First In First Out*), RED (*Robust Earliest Deadline*) ou WFQ (*Weighted Fair Queuing*). La couche liaison ou MAC (*Medium Access Control*), offre un service de transmission point à point entre les différents nœuds du réseau. Cette couche implémente plusieurs techniques d'accès dont le CSMA (*Carrier Sense Multiple Access*), MACA (*Multiple*

Access with Collision Avoidance), TSMA (*Time Spread Multiple Access*) ainsi que les normes IEEE 802.11 pour le sans fil et IEEE 802.3 pour les réseaux câblés. La couche physique, quant à elle, permet un accès brut au canal de communication. Outre les protocoles offerts à travers les différentes couches, le simulateur implante un grand nombre de modèles de communication radio, de propagation ou de mobilité.

La modification d'un protocole existant ou l'ajout d'un nouveau protocole se fait à même les fichiers sources de QualNet. Une fois l'édition terminée, il est nécessaire de compiler l'ensemble des fichiers en un exécutable nommé *qualnet.exe*. Cette compilation peut se faire à l'aide de n'importe quel compilateur capable d'interpréter le langage de programmation C. Une fois la compilation réussie, il est possible de simuler différents scénarios en spécifiant les caractéristiques de simulation désirées. Les résultats des simulations sont donnés sous forme textuelle.

4.2.2 Détails d'implémentation

Dans cette section, nous nous intéressons principalement aux détails d'implémentation de notre protocole. Ainsi, nous commençons par décrire la stratégie d'implémentation dans le simulateur et nous terminons la section en présentant les structures de données adoptées ainsi que les fonctions utilisées dans l'implantation de notre protocole.

Stratégie d'implémentation dans QualNet Simulator

Notre implémentation porte majoritairement sur la couche application. Contrairement à QualNet qui est programmé en C, notre implémentation est effectuée en C++, qui permet une approche orientée objet, tout en gardant la compatibilité avec le C. Cette approche encourage une conception claire et efficace. Ainsi, l'implémentation est effectuée à l'aide de différentes classes. Comme l'implémentation d'ORION n'est pas disponible, elle a dû être effectuée également. Cependant, grâce à leur fonctionnement similaire, plusieurs structures de données sont partagées entre les deux algorithmes.

L'implémentation effectuée pour chaque protocole comprend les deux algorithmes, soit l'algorithme de recherche et l'algorithme de transfert. Pour l'algorithme de recherche, uniquement la recherche par mots clefs a été implémentée, puisque l'objectif est de comparer les performances des transferts.

Dans les prochaines sections, nous verrons les implémentations des différentes

parties des protocoles.

La base de fichiers

Pour implémenter la base de fichiers, deux classes ont été développées, soit la classe *CFile* et la classe *CRepository.*

La classe *CFile* représente un fichier dans la base de fichiers du mobile et est partagée par les deux algorithmes implémentés. Les principaux attributs de cette classe sont :

- *mComplete* : il s'agit d'un booléen qui est mis à vrai si le fichier est complet et à faux si non ;
- *mPath* : il s'agit d'une chaîne de caractères représentant le chemin du fichier sur le disque dur ;
- *mName* : il s'agit d'une chaîne de caractères représentant le nom du fichier sur le disque dur ;
- *mSize* : il s'agit d'un entier représentant la taille du fichier ;
- *mNbPieces* : il s'agit d'un entier représentant le nombre de morceaux dans le fichier ;
- *mPieceSize* : il s'agit d'un entier représentant la taille d'un morceau ;
- *mMD5PieceSums* : il s'agit de la liste des signatures des morceaux du fichier ;
- *mMD5Sum* : il s'agit de la signature du fichier.

Les principales méthodes de la classe sont :

- *load()* : permet de charger les informations requises à partir du fichier sur le disque dur ;
- *create()* : crée un nouveau fichier sur le disque dur ;
- *getData()* : retourne les données lues du fichier ;
- *writeData()* : écrit des données dans le fichier.

La classe CRepository représente la base de fichiers du mobile et est partagée par les deux algorithmes implémentés. Les principaux attributs de cette classe sont :

- *mPath* : il s'agit d'une chaîne de caractères repésentant le chemin sur le disque dur de la base de données ;
- *mFileList* : il s'agit d'une mappe permettant de retrouver l'adresse d'un fichier à partir de sa signature.

Les principales méthodes de la classe sont :

- *getFileFromMD5()* : retourne l'adresse d'un fichier à partir de sa signature ;
- *addNewFile()* : ajoute un nouveau fichier dans la base, en le créant sur le disque

dur ;

- *addExistingFile()* : ajoute dans la base un fichier déjà existant sur le disque dur.

Les messages

Tous les messages utilisés dans les deux algorithmes dérivent d'une classe générique, soit la classe *CMsgBase*. Pour l'implémentation d'ORION, quatre classes de messages ont été implémentées, soit *CMsgFReq*, *CMsgFRep*, *CMsgDReq* et *CMsgDRep*. Pour l'implémentation de la solution proposée, deux autres classes ont été rajoutées, soit *CMsgSReq* et *CMsgSRep*.

Comme les messages doivent être envoyés sur le réseau, chaque objet contenant un message doit pouvoir être converti en une chaîne de caractères (serialization), qui pourra alors être envoyée sur le réseau. De plus, il doit pouvoir effectuer l'action inverse, soit de convertir une chaîne de caractères en un objet.

La classe abstraite *CMsgBase* représente la classe de base pour tout message. L'unique attribut qu'elle contient est un entier qui indique le type du message. Les principales méthodes de cette classe sont des méthodes virtuelles pures qui doivent être définies dans les classes dérivées :

- *getSerSize()* : cette méthode retourne la taille de la chaîne de caractères nécessaire pour contenir le message ;
- *serialize()* : cette méthode écrit sur une chaîne de caractères le message ;
- *unserialize()* : cette méthode effectue le travail inverse de la précédente, soit lire les informations contenues dans la chaîne de caractères et initialiser les attributs avec ces valeurs.

La classe *CMsgFReq* représente un message de type requête de fichiers. Ses principaux attributs sont :

- *mSrc* : l'adresse du nœud source (le nœud requête) ;
- *mLastHop* : l'adresse du dernier nœud à avoir relayé le message ;
- *mSeq* : le numéro de séquence de la requête ;
- *mTTL* : le nombre de sauts maximal permis ;
- *mNbHops* : le nombre de sauts effectués par le message ;
- *mMD5Digest* : la signature du fichier recherché ;

Cette classe implémente les méthodes virtuelles pures héritées et ne définit aucune nouvelle méthode.

La classe *CMsgFRep* représente un message de type réponse d'une requête de fichiers. Ses principaux attributs sont :

- *mDst* : l'adresse de la destination (le nœud requête) ;
- *mSrc* : l'adresse de la source (le nœud réponse) ;
- *mLastHop* : l'adresse du dernier nœud à avoir relayé le message ;
- *mSeq* : le numéro de séquence de la requête ;
- *mTTL* : le nombre de sauts maximal permis ;
- *mNbHops* : le nombre de sauts effectués par le message ;
- *mFileSize* : la taille du fichier ;
- *mMD5Digest* : la signature du fichier ;
- *mFilename* : le nom du fichier ;
- *mBitField* : un vecteur de booléen contenant les drapeaux du bitfield. Si le fichier est complet, ce vecteur est vide.

Cette classe implémente les méthodes virtuelles pures héritées et ne définit aucune nouvelle méthode.

La classe *CMsgDReq* représente un message de type requête de morceau. Ses principaux attributs sont :

- *mSrc* : l'adresse du nœud source (le nœud requête) ;
- *mLastHop* : l'adresse du dernier nœud à avoir relayé le message ;
- *mTTL* : le nombre de sauts maximal permis ;
- *mOffset* : l'index du morceau désiré ;
- *mMD5Digest* : la signature du fichier.

Cette classe implémente les méthodes virtuelles pures héritées et ne définit aucune nouvelle méthode.

La classe *CMsgDRep* représente un message de type réponse d'une requête de morceau. Ses principaux attributs sont :

- *mSrc* : l'adresse du nœud source (le nœud réponse) ;
- *mDest* : l'adresse du nœud destination (le nœud requête) ;
- *mLastHop* : l'adresse du dernier nœud à avoir relayé le message ;
- *mTTL* : le nombre de sauts maximal permis ;
- *mOffset* : l'index du morceau désiré ;
- *mMD5Digest* : la signature du fichier.
- *mBuf* : une chaîne de caractères contenant les données lues dans le fichier ;
- *mBitField* : un vecteur de booléen contenant les drapeaux du bitfield. Si le

fichier est complet, ce vecteur est vide.

Cette classe implémente les méthodes virtuelles pures héritées et ne définit aucune nouvelle méthode.

La classe *CMsgSReq* représente un message de type requête des signatures des morceaux. Ses principaux attributs sont :

- *mSrc* : l'adresse du nœud source (le nœud requête) ;
- *mLastHop* : l'adresse du dernier nœud à avoir relayé le message ;
- *mTTL* : le nombre de sauts maximal permis ;
- *mMD5Digest* : la signature du fichier.

Cette classe implémente les méthodes virtuelles pures héritées et ne définit aucune nouvelle méthode.

La classe *CMsgSRep* représente un message de type réponse d'une requête des signatures des morceaux. Ses principaux attributs sont :

- *mSrc* : l'adresse du nœud source (le nœud réponse) ;
- *mDest* : l'adresse du nœud destination (le nœud requête) ;
- *mLastHop* : l'adresse du dernier nœud à avoir relayé le message ;
- *mTTL* : le nombre de sauts maximal permis ;
- *mMD5Digest* : la signature du fichier.
- *mMD5Sums* : le vecteur contenant les signatures.

Cette classe implémente les méthodes virtuelles pures héritées et ne définit aucune nouvelle méthode.

Les tables

L'implémentation de chacun des protocoles contient les tables nécessaires à leur fonctionnement. Pour ORION, on retrouve la table de réponses et la table de fichiers, telles que spécifiées dans le protocole. Pour le protocole proposé, on retrouve la table de routage et la table de transferts.

La classe CRepTable représente la table de routage du protocole proposé. Son fonctionnement est similaire à celui d'ORION, avec comme différence la possibilité de mémoriser plusieurs prochains sauts pour une destination donnée.

La classe CTransferTable représente la table de transfert du protocole proposé. Elle permet, pour un fichier donné, de mémoriser les sources ainsi que le progrès du transfert. Ainsi, pour chaque fichier contenu dans la table, on retrouve les attributs suivants :

- *mNodes* : la liste des sources (complètes ou partielles) ;
- *mFile* : un pointeur vers le fichier dans la base ;
- *mCurPos* : le dernier morceau demandé à la source ;
- *mNbPieces* : le nombre de morceaux reçus ;
- *mMaxPieces* : le nombre de morceaux dans le fichier ;
- *mStartTime* : le temps de démarrage du transfert.

Les classes principales

Chacun des protocoles est implémenté dans un classe distincte. On retrouve donc deux classes, soient CORION et CGAK. La classe CORION implémente le protocole ORION. Ses principaux attributs sont :

- *mRepository* : la base de fichiers ;
- *mNode* : un pointeur vers le nœud (fourni par QualNet) ;
- mLocalAddress : l'adresse locale ;
- *mSeq* : le dernier numéro de séquence utilisé ;
- *mRepTable* : la table de réponses ;
- *mFileTable* : la table de fichiers ;

Les principales méthodes sont :

- *startRequest()* : démarre une recherche pour une signature donnée ;
- *startDownload()* : démarre un transfert pour une signature donnée ;

La classe CSP implémente la solution proposée. Ses principaux attributs sont :

- *mRepository* : la base de fichiers ;
- *mNode* : un pointeur vers le nœud (fourni par QualNet) ;
- *mLocalAddress* : l'adresse locale ;
- *mSeq* : le dernier numéro de séquence utilisé ;
- *mRepTable* : la table de réponses ;
- *mTransferTable* : la table de transferts.

Les principales méthodes sont :

- startRequest() : démarre une recherche pour une signature donnée ;
- startDownload() : démarre un transfert pour une signature donnée.

4.2.3 Plan d'expériences et de simulation

Cette section présente le plan d'expériences qui sera utilisé pour évaluer les performances de la solution proposée dans ce mémoire vis-à-vis de celles d'ORION. Outre la configuration des scénarios et des différents éléments de simulation, le plan d'expérience définit un certain nombre d'indices utilisés pour évaluer la performance de notre protocole sous certaines conditions.

Configuration de la simulation

Dans le cadre de notre série de simulations, nous considérons un réseau mobile ad hoc distribué uniformément sur une surface de 1250m * 1250m, composé de 28 nœuds mobiles. Toutes les unités mobiles utilisent une interface de communication sans fil de type 802.11b en mode ad hoc. Ainsi, en mode ad hoc, les unités mobiles du réseau communiquent point-à-point et ne requièrent donc pas de station de base ou de point d'accès centralisé. L'utilisation de la même technologie d'accès au médium permet d'avoir une même portée de transmission, soit 50 mètres, et un même débit à 2 Mbps. Il est à noter que les signaux sont propagés dans le réseau selon le modèle en espace libre. Aucun modèle de mobilité n'est choisi pour les scénarios, puisque les mécanismes de réparation de routes ne sont pas implémentés.

Les protocoles présents au niveau de la couche réseau des mobiles sont IP et AODV. Tous les nœuds exécutent également le protocole de partage au niveau de leur couche application et participent donc à son fonctionnement. Les bases de fichiers de tous les nœuds sont vides, sauf pour le nœud 1 qui possède le fichier Video.avi. Ainsi, le nœud 1 est une source complète de ce nœud, les autres nœuds étant des clients potentiels. L'emplacement de la source et des clients est aléatoire pour chaque simulation. Afin d'évaluer l'évolutivité de notre protocole, nous faisons varier le nombre de clients de 1 à 5, avec un incrément de 1. Le nombre de morceaux qui constituent le fichier est également varié de 25 à 50, à 100 et à 200. Tous les transferts démarrent simultanément. La taille des morceaux est de 32 kilo octets, chaque morceau pouvant être envoyé en un seul paquet IP. Chaque paquet est injecté avec un certain délai aléatoire compris en 0 et 50 millisecondes, permettant ainsi de réduire le taux de collisions. Le délai maximal d'attente d'une réponse est de 1 seconde. Après ce délai, on considère la requête perdue et elle doit être retransmise.

Dans le but de minimiser les effets imputables au hasard, chaque scénario S de

simulation, c'est-à-dire un couple S = (nombre de clients, nombre de morceaux), sera reproduit 10 fois. Ainsi, les résultats obtenus correspondront à des moyennes sur des séries de simulations. Il est également à noter que chaque scénario sera simulé pour une durée de 20 minutes, ce qui est largement suffisant pour la complétion de tous les transferts.

Le Tableau 4.2 résume la configuration des principaux paramètres utilisés durant les simulations. Il est à noter que les paramètres dynamiques, qui changent d'une simulation à l'autre, sont surlignés en gris.

Indices de performance

Afin de déterminer la performance des deux protocoles de partage de fichiers, nous avons utilisé les 4 métriques suivantes :

- *Le temps moyen de transfert.* Il s'agit de la moyenne de tous les temps de transfert des clients. Cette métrique est considérée comme la plus importante car il s'agit de la métrique perçue par l'usager ;
- *Le débit total envoyé.* Il s'agit de la somme des débits envoyés dans le réseau par chaque nœud ;
- *Le nombre total de paquets envoyés.* Il s'agit de la somme de tous les paquets envoyés dans le réseau par chaque nœud ;

4.2.4 Analyse des résultats de simulation

À ce stade, les simulations visant à évaluer les performances des protocoles ont été réalisées. Dans cette section, nous présenterons les résultats obtenus suite à ces simulations et les comparerons aux résultats obtenus pour les deux protocoles. Étant donné la grande quantité de résultats générés à partir de notre plan d'expérience, nous avons choisi de montrer ici un sous-ensemble des résultats obtenus. Pour chaque scénario simulé, seul un sous ensemble du réseau sera considéré, soient les nœuds clients, le nœud source ainsi que les nœuds intermédiaires des routes les reliant. Les autres nœuds ne participent nullement aux transferts et seront ignorés. De plus, le fichier utilisé ne modifie pas le comportement des protocoles. Ainsi, seuls les transferts avec le fichier de 100 morceaux seront montrés. Les résultats présentés dans cette section sont donc considérés comme tendanciels, les autres n'apportant pas d'éléments nouveaux dans l'évaluation de notre solution.

TABLEAU 4.2 Configuration des paramètres de simulation

Paramètre	**Valeur**(s)
Taille de l'aire de simulation (m)	1000 * 1000
Nombre de nœuds	20
Distribution des nœuds	Uniforme
Protocole de couche MAC	802.11b
Protocole de couche réseau	IP et AODV
Débit (Mbps)	2
Type d'antenne	Omnidirectionnelle
Portée de transmission (m)	50
Modèle de propagation	Espace libre
Modèle de mobilité	Aucun
Nombre de sources complètes	1
Emplcacement de la source	Aléatoire
Emplacement des clients	Aléatoire
Nombre de clients	1, 2, 3, 4, 5
Nombre de paquets dans le fichier	25, 50, 100, 200
Taille des morceaux	32 kilo octets
Délai d'injection des paquets	Aléatoire dans [0 ; 50] ms
Délai maximal d'attente	1 seconde

Un client

Lorsque les scénarios contenant un seul client sont simulés, on remarque que la seule métrique qui influence la performance des protocoles est la longueur de la route entre ce client et la source, donc le nombre de nœuds intermédiaires qui jouent le rôle de routeurs. La Figure 4.2 montre le temps de transfert des deux protocoles dans le scénario avec un seul client, en fonction du nombre de sauts qui le séparent de la source. On remarque donc que la performance des deux protocoles est comparable. De plus, les Figures 4.3 et 4.4 montrent respectivement le débit total et le nombre de paquets envoyés dans le réseau. Comme pour le temps de transfert, les résultats obtenus sont similaires pour les deux résultats. Les résultats expérimentaux avec un seul client sont donc en concordance avec les résultats obtenus avec le modèle analytique. En effet, avec un seul client, la solution proposée se comporte comme ORION avec comme seule différence l'échange des signatures, étape effectuée une seule fois au début du transfert et négligeable par rapport au reste des données transférées.

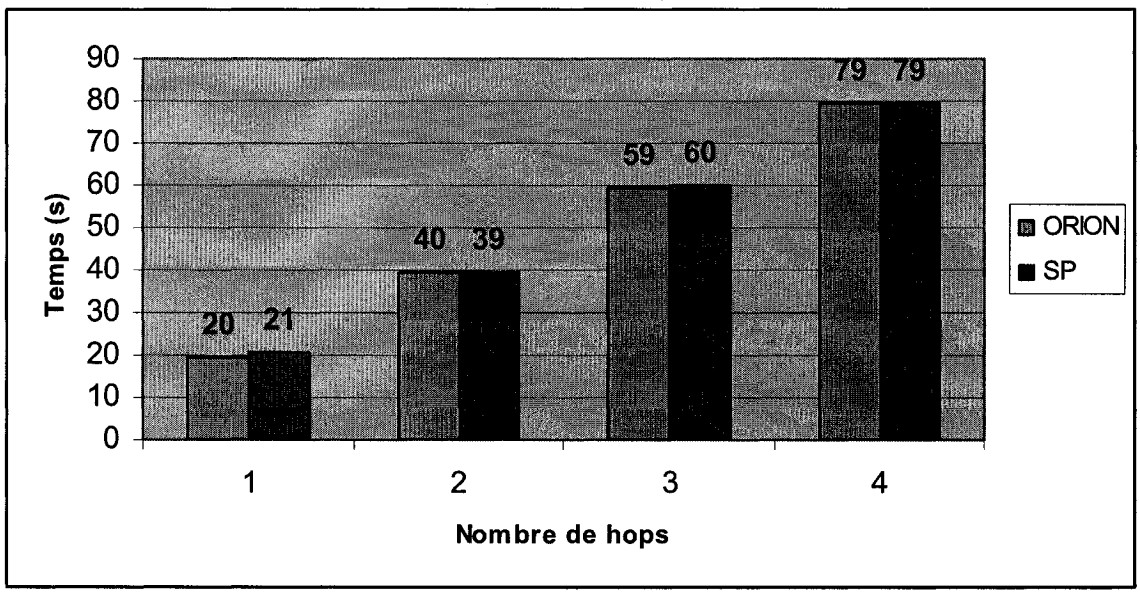


FIGURE 4.2 Temps de transfert pour 1 client

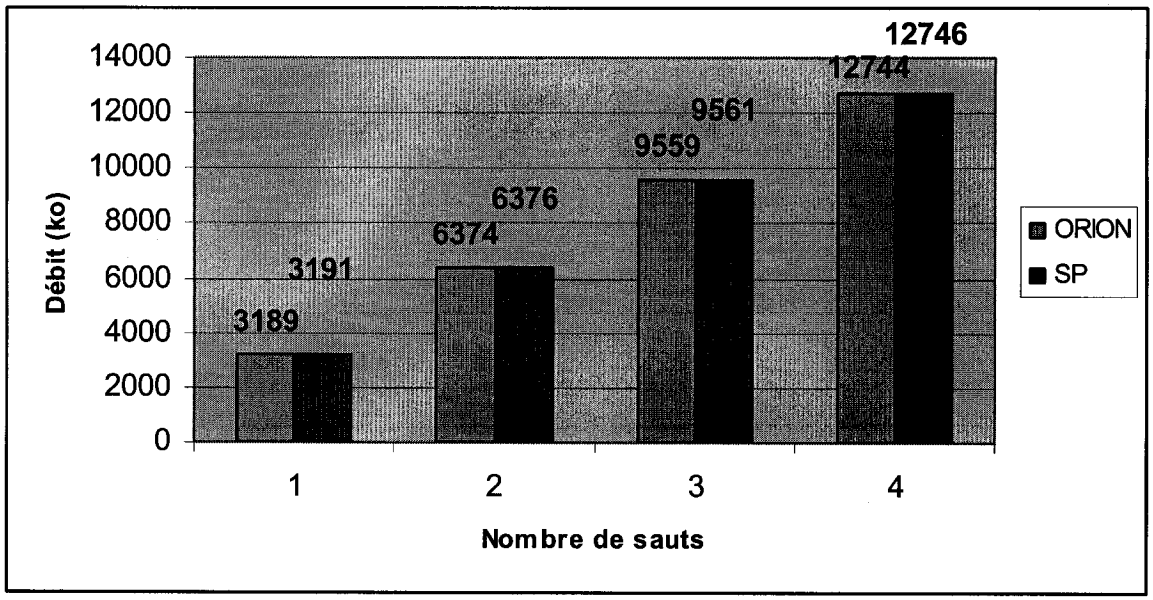


FIGURE 4.3 Débit total envoyé pour 1 client

Deux clients

Lorsque les scénarios contenant deux clients sont simulés, on remarque que le positionnement des clients influence directement la performance des protocoles. En effet, il faut distinguer plusieurs cas. La Figure 5 montre un scénario possible avec deux clients. Dans ce scénario, la source complète est la meilleure source pour les deux clients. Seul le scénario où les clients sont séparés par un seul saut de la source sera étudié.

La Figure 4.6 montre les résultats obtenus lorsque ce scénario est simulé. Comme dans le scénario à un client, la performance des deux protocoles est similaire. En effet, comme pour chaque client la meilleure source est la source complète tout au long des transferts, la solution proposée se comporte comme ORION et on obtient les mêmes résultats. Il est à noter qu'on peut augmenter la taille des routes entre les clients et la source et que, tant que la source complète est la meilleure source pour les deux clients, les résultats obtenus seront semblables pour les deux solutions.

Afin de pouvoir observer l'efficacité de la solution proposée, considérons le scénario B montré à la Figure 4.7. Dans ce scénario, le client 1 est une meilleure source pour le client 2 que la source complète. De plus, il est le noeud intermdiaire entre la source

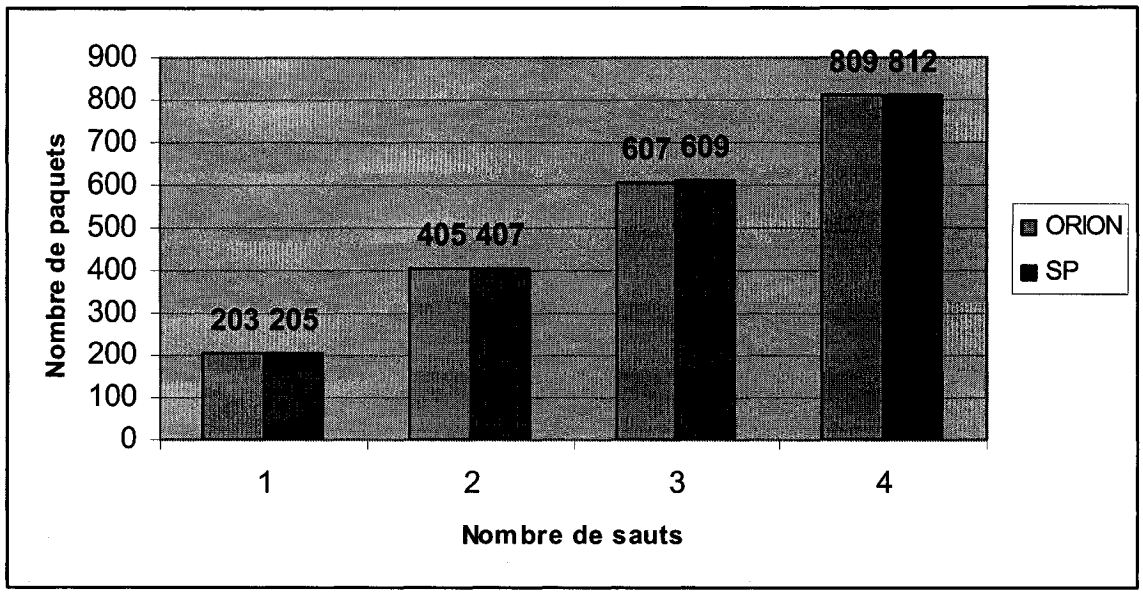


FIGURE 4.4 Nombre de paquets envoyés pour 1 client

et le client 2.

La Figure 4.8 montre le temps de transfert par client. On remarque que la solution proposée offre de bien meilleures performances, soit une réduction de 57.5% du temps de transfert du client 1 et de 37.5% du temps de transfert du client 2. Ces réductions viennent du fait que, dans la solution proposée, le client 3 est désormais desservi par le client 2, source partielle, alors qu'avec ORION le client 3 est desservi par la source complète. Ainsi, avec la solution proposée, le client 1 ne joue plus le rôle de routeur pour le transfert du client 2, ce qui lui permet d'utiliser exclusivement la

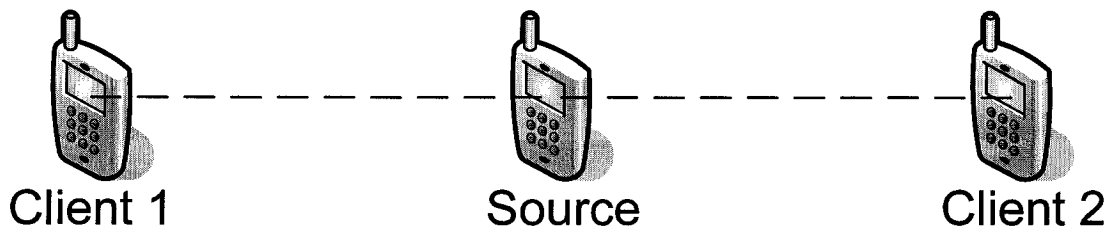


FIGURE 4.5 Scénario A avec 2 clients

route entre lui et la source complète pour son transfert. Avec ORION, cette route doit être partagée entre les transferts des deux clients. De plus, avec la solution proposée, on réduit considérablement le débit envoyé au niveau de la source complète, comme montré à la Figure 4.9. Cette réduction, qui est de 57.5%, vient du fait que la source complète ne dessert qu'un seul client désormais. De plus, on réduit le nombre de paquets envoyés par le client 1 car ce dernier ne relaye plus les requêtes du client 2 à la source complète, comme montré à la Figure 4.10.

Il est également intéressant de noter que le nombre de paquets perdus dû à la collision est également réduit considérablement, passant de 12 (ORION) à 4 (Solution proposée), pour une réduction de 67%.

Trois clients et plus

Pour les scénarios composés de trois clients ou plus, nous montrerons un cas intéressant, soit celui de la Figure 4.11.

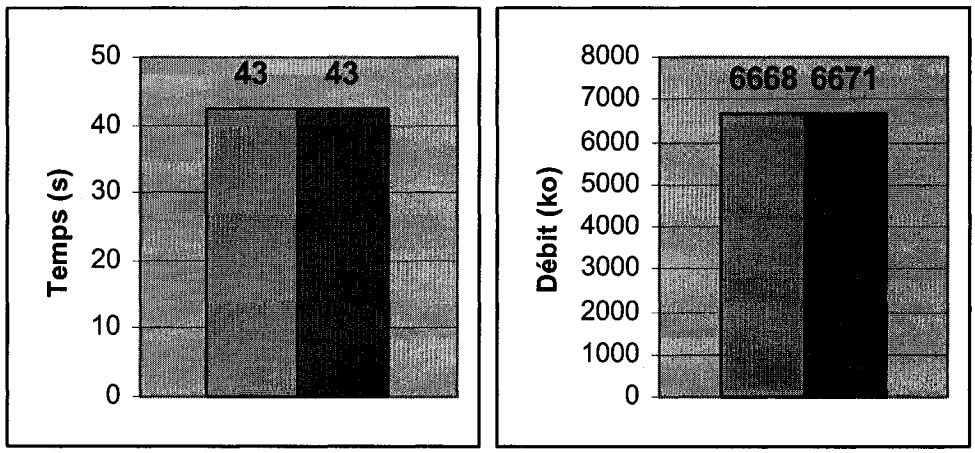
Seuls les résultats au niveau du temps de transfert sont montrés, soit à la Figure 4.12.

L'amélioration du temps de transfert au niveau des clients 2 et 3 n'est pas surprenante, puisqu'il s'agit d'un scénario semblable au précédent. Cependant, l'amélioration du temps de transfert du client 1 (réduction de 40%) peut paraitre, à première vue, surprenante. L'explication est simple, avec la solution proposée, la source complète ne dessert que deux clients, contrairement à trois avec ORION. Or, cette diminution de la charge diminue la congestion au niveau de la source, ce qui engendre de plus petits délais lors des traitements des requêtes. Ainsi, même si la source est la même au niveau du client 1 pour les deux protocoles, la performance de la solution proposée est meilleure car la congestion au niveau de la source est réduite grâce à l'élimination des requêtes du client 3, qui sera desservi par le client 2.

4.3 Synthèse des résultats obtenus

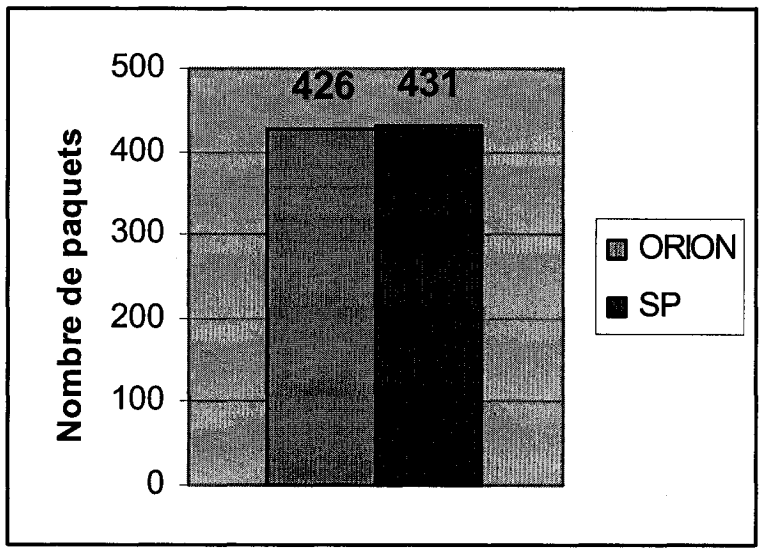
Suite aux analyses des résultats obtenus avec le modèle analytique et le modèle expérimental, il est maintenant possible de synthétiser les résultats obtenus, de situer la solution proposée, et de démontrer son mérite. Les résultats obtenus par notre protocole sont, dans le pire cas, aussi bons que ceux obtenus avec ORION. Dans un scénario où des clients transfèrent le même fichier simultanément, la solution proposée

permet dans la plupart des cas de réduire considérablement le temps de transfert des clients. Elle permet également de réduire le débit envoyé dans le réseau. En plus de ces améliorations, la solution proposée offre une meilleure sécurité face aux transferts erronés puisqu'elle permet de détecter les erreurs au niveau des morceaux, et non pas au niveau du fichier, ce qui réduit énormément le nombre de données retransmises lors d'erreurs. De plus, la solution proposée est plus robuste face aux pannes. En effet, si à un moment donné toutes les sources complètes pour un fichier donné disparaissent, avec ORION les nœuds se retrouvent bloqués, donc tous ces transferts restent non complétés. Avec la solution proposée, les clients sont des sources partielles et peuvent s'échanger des morceaux entre eux. De plus, si chaque morceau se retrouve au moins une fois dans le réseau, le fichier peut être reconstitué en entier et tous les transferts peuvent alors être complétés.



(a) Temps de transfert

(b) Débit envoyé



(c) Nombre de paquets envoyés

FIGURE 4.6 Résultats du scénario A pour 2 clients

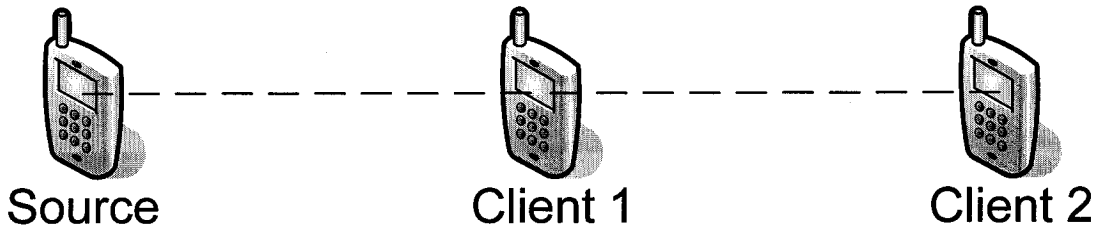


FIGURE 4.7 Scénario B avec 2 clients

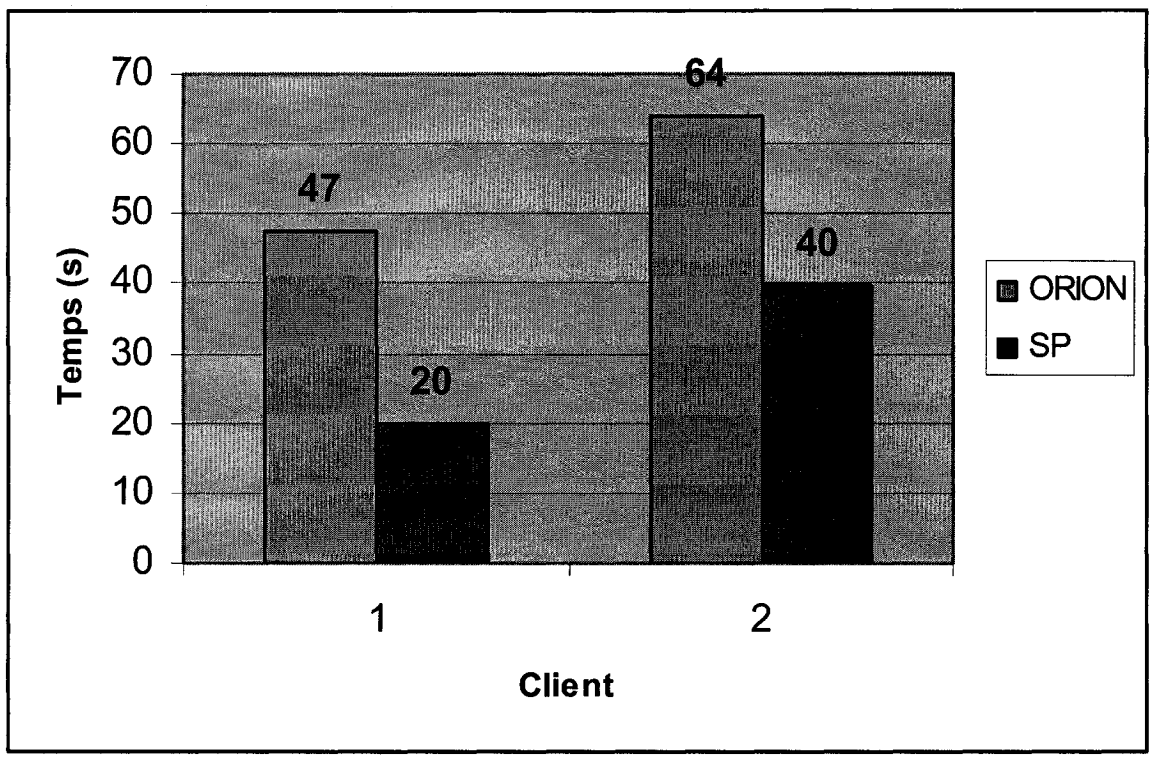


FIGURE 4.8 Temps de transfert des clients (scénario B pour 2 clients)

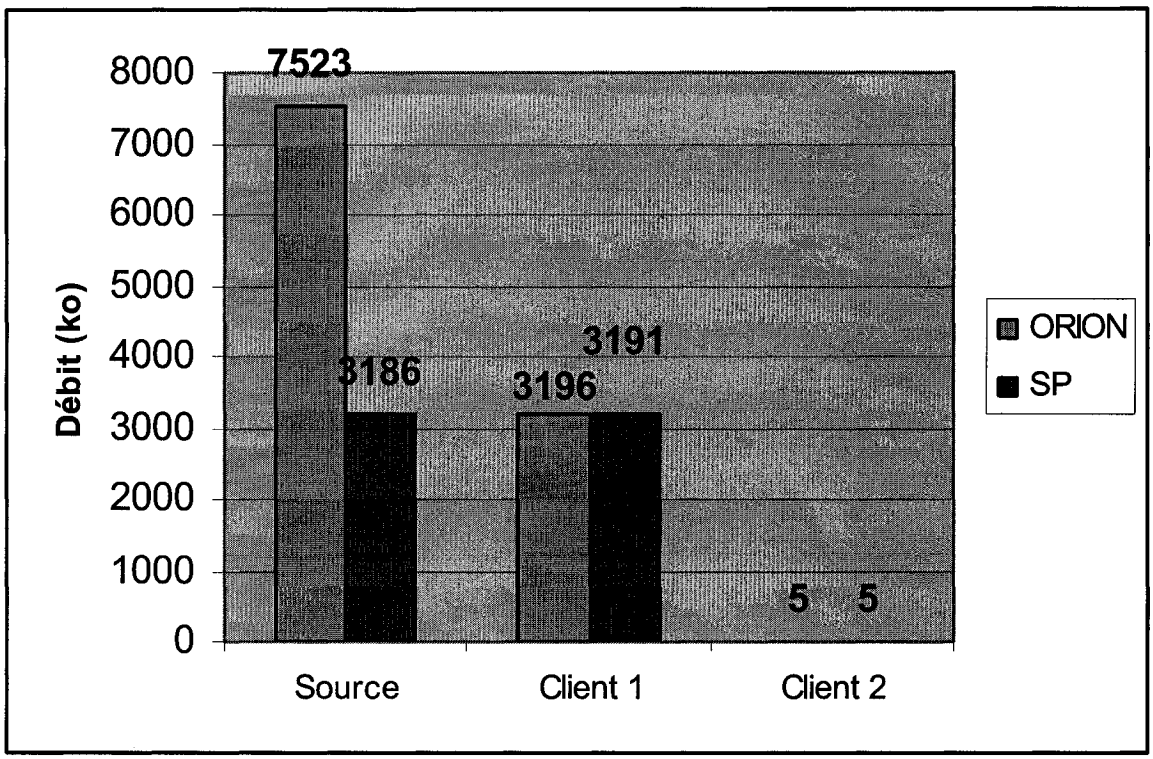


FIGURE 4.9 Débit envoyé par nœud (scénario B avec 2 clients)

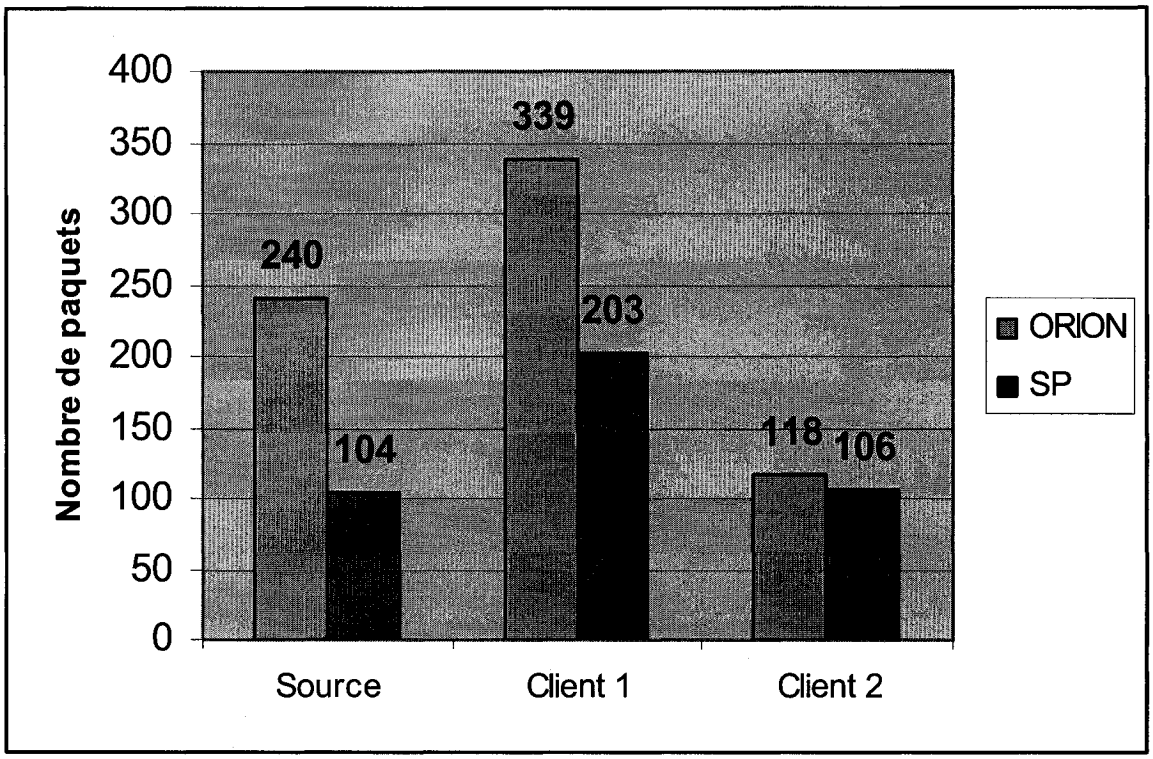


FIGURE 4.10 Nombre de paquets envoyés par nœud (scénario B avec 2 clients)

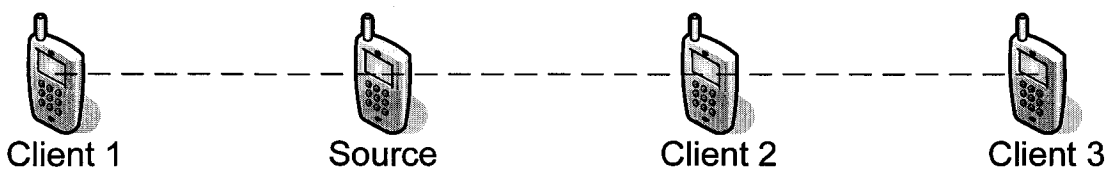


FIGURE 4.11 Scénario avec 3 clients

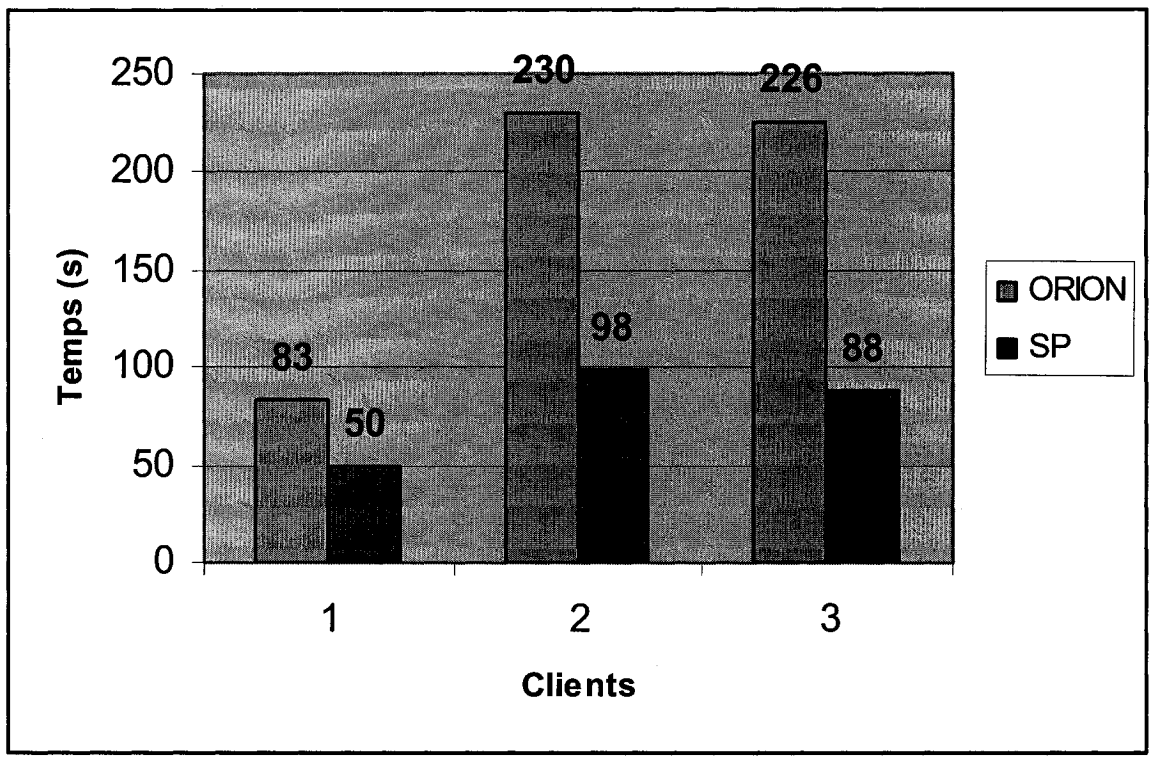


FIGURE 4.12 Temps de transfert du scénario avec 3 clients

CHAPITRE 5

CONCLUSION

Dans ce chapitre, nous réviserons les points importants du projet de recherche. Lors de la revue de littérature, nous avons exploré les différents aspects reliés au routage, ainsi que les différents algorithmes existants pour les réseaux ad hoc. De plus, nous avons exploré les différents aspects reliés aux réseaux de partage pair à pair ainsi que les différents protocoles existants pour les réseaux filaires et les réseaux ad hoc. Lors des deux chapitres suivants, nous avons énonceé notre solution et les résultats, respectivement.

Ce chapitre vient clore les démarches entreprises dans ce projet. Nous ferons une synthèse des travaux accomplis et présenterons l'ensemble des limitations de notre protocole. Enfin, nous discuterons des améliorations possibles à notre protocole, qui pourraient faire l'objet de travaux futurs.

5.1 Synthèse des travaux

La contribution majeure de ce travail est de proposer un nouveau protocole de partage de fichiers pair à pair pour les réseaux mobiles ad hoc. Ce protocole s'inspire d'ORION, protocole déjà existant. Le protocole proposé se distingue d'ORION par de meilleures performances lors des transferts de fichiers. Ceci est dû majoritairement à un mécanisme ajouté, introduit par Bittorrent. Ce mécanisme permet aux clients qui téléchargent un fichier de le desservir également, comme source partielle. Ainsi, au fur et à mesure qu'un nœud obtient des morceaux d'un fichier, il peut alors desservir ces morceaux à d'autres clients qui les demanderaient. Ce mécanisme permet de réduire le nombre de paquets dans le réseau puisqu'il augmente le nombre de sources pour un fichier donné, ce qui permet à des clients de trouver des sources plus proches, réduisant ainsi la taille des routes. L'ajout de ce mécanisme permet également aux clients de s'échanger les morceaux même si aucune source complète n'est présente dans le réseau. De plus, si tous les morceaux du fichier se trouvent au moins une

fois dans le réseau, les clients peuvent reconstituer tout le fichier et terminer leurs transferts et ce, sans l'aide de source complète.

Une autre contribution apportée se trouve au niveau de la sécurité. Avec le protocole original, une erreur de transfert (qui peut être le résultat d'un problème matérielle ou l'œuvre d'un nœud malicieux) n'est détectée que lorsque le fichier est complètement téléchargé. Ainsi, si le nœud réalise que le transfert n'est pas réussi, il ne peut identifier le morceau erroné et doit recommencer le transfert complètement. Avec la solution proposée, le nœud peut effectuer la vérification des données morceau par morceau, ce qui permet une retransmission partielle lors d'un problème de transfert.

Finalement, le protocole proposé permet de gérer les transferts simultanés en série. Ainsi, pour éviter les congestions et donc les pertes de paquets, le protocole considère l'ensemble des transferts comme un seul transfert. Il permet également d'ajuster la priorité de certains transferts par rapport à d'autres, permettant ainsi de réserver davantage de bande passante pour des transferts plus importants.

5.2 Limitations des travaux

Les résultats obtenus montrent bien que les performances de notre protocole sont meilleures que celles d'ORION. Cependant, les simulations effetuées comportent une limitation, soit la mobilité. En effet, il serait intéressant d'implémenter les mécanismes de réparation de routes dans les deux protocoles et d'effectuer de nouvelles simulations avec des réseaux mobiles. Nous ne croyons cependant pas que les résultats divergeraient, car le mécanisme de réparation est relativement semblable dans les deux protocoles.

5.3 Améliorations futures

Cette section présente les différentes améliorations possibles à notre solution, pouvant faire l'objet de travaux futurs.

5.3.1 Mécanisme de mise en antémémoire

Les réseaux de partage de fichier pair à pair ne garantissent d'aucune façon la disponibilité d'un fichier. Dans le réseau ad hoc, ce problème est accentué avec la

mobilité. Pour cette raison, il serait intéressant d'offrir un mécanisme de mise en antémémoire pour les fichiers. Un tel mécanisme est proposé par Rajagopalan et Shen (2006). Pour chaque nouveau fichier dans le réseau, des sources caches peuvent être créées automatiquement, augmentant ainsi le nombre de sources disponibles pour ce fichier. La création de ces sources doit être automatique (sans l'intervention de l'usager) et doit être bien gérée afin de ne pas surcharger le réseau. De plus, il est important de bien choisir les nœuds qui vont servir de sources caches afin de maximiser la performance des transferts éventuels de ce fichier.

5.3.2 Gestionnaire de congestion

Une autre amélioration possible est l'ajout d'un gestionnaire de congestion. En effet, si tous les nœuds lancent des transferts simultanément, le réseau risque d'être saturé, ce qui engendre d'énormes délais dus à l'augmentation du taux de collisions. Il serait donc intéressant que l'application puisse détecter le niveau de congestion du nœud et prendre des décisions en conséquence. Ainsi, lorsqu'un nœud se considère congestionné, il peut arrêter de traiter les requêtes de fichiers qu'il reçoit, jusqu'à que son niveau de congestion ne baisse sous un certain seuil. Cette technique obligerait les transferts de passer par des routes composées de nœuds capables de les traiter. Une autre méthode serait de modifier la métrique des routes afin de prendre en compte non seulement le nombre de sauts, mais également la congestion sur chacun des nœuds intermédiaires. Le problème engendré cependant par ces techniques est le même problème retrouvé dans les protocoles de routage, soit l'incitation des nœuds à la participation. Il faut donc que l'implémentation du gestionnaire de congestion tienne compte également de ce problème.

5.3.3 Modification de la logique d'envoi

Une autre amélioration possible est le transfert de la logique d'envoi du client vers la source. Dans le protocole actuel, c'est le client qui choisit aléatoirement le morceau qu'il désire recevoir. Il serait intéressant de modifier ce comportement en transférant le choix du morceau vers la source. Ainsi, on permettrait aux sources de mieux répartir les morceaux augmentant ainsi les probabilités qu'un client trouve une source partielle intéressante. De plus, dans le cas où toutes les sources complètes disparaissent, on augmenterait les probabilités que tous les morceaux du fichier se

trouvent parmi les sources partielles. Pour implémenter cette modification, on peut s'inspirer du travail déjà effectué pour le protocole Bittorrent.

Références

CHEN, T.-W. et GERLA, M. (1998). Global state routing : a new routing scheme for ad-hoc wireless networks. *IEEE International Conference on Communications (ICC)*. Atlanta, USA, vol. 1, 171–175.

HAAS, Z. J. (1997). A new routing protocol for the reconfigurable wireless networks. *IEEE International Conference on Communications (ICC)*. IEEE, IEEE, San Diego, California, USA, vol. 2, 562–566.

JACQUET, P., MÜHLETHALER, P., CLAUSEN, T., LAOUITI, A., QAYYUM, A. et VIENNOT, L. (2001). Optimized link state routing protocol for ad hoc networks. *Proceedings of the 5th IEEE Multi Topic Conference (INMIC)*. Lahore, Pakistan, 62–68.

JOHNSON, D. (1994). Routing in ad hoc networks of mobile hosts. *Workshop on Mobile Computing Systems and Applications*. Santa Cruz, CA, U.S., 158–163.

JOHNSON, D. B. et MALTZ, D. A. (1996). Dynamic source routing in ad hoc wireless networks. T. Imielinski et H. Korth, éditeurs, *Mobile Computing*, Kluwer Academic Publishers, vol. 353, chapitre 5. 153–181.

KARP, B. et KUNG, H. T. (2000). Gpsr : Greedy perimeter stateless routing for wireless networks. *6th Annual International Conference on Mobile Computing and Networking, (MobiCom)*. Boston, Massachusetts, USA, 243–254.

KLEMM, A., LINDEMANN, C. et WALDHORST, O. P. (2003). A special-purpose peer-to-peer file sharing system for mobile ad hoc networks. *Vehicular Technology Conference*. Orlando, Floride, USA, vol. 4, 2758–2763.

LINDEMANN, C. et WALDHORST, O. (2002). A distributed search service for peer-to-peer file sharing in mobile applications. *Second International Conference on Peer-to-Peer Computing*. Linköping, Suède, 73–80.

OLIVEIRA, L. B., SIQUEIRA, I. G. et LOUREIRO, A. A. F. (2003). Evaluation of ad-hoc routing protocols under a peer-to-peer application. *IEEE Wireless Communications and Networking Conference (WCNC'03)*. New Orleans, USA, 1143–1148.

PAPADOPOULI, M. et SCHULZRINNE, H. (2000). Seven degrees of separation in mobile ad hoc networks. *IEEE Global Telecommunications Conference 2000 (GLOBECOM '00)*. San Fransisco, USA, vol. 3, 1707–1711.

PEI, G., GERLA, M. et CHEN, T.-W. (2000). Fisheye state routing : a routing scheme for ad hoc wireless networks. *IEEE International Conference on Communications (ICC)*. New Orleans, USA, vol. 1, 70–74.

PERKINS, C. E. et BHAGWAT, P. (1994). Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM Conference on Communications Architectures, Protocols and Applications*. ACM, ACM, London, UK, 234–244.

PERKINS, C. E. et ROYER, E. M. (1999). Ad-hoc on-demand distance vector routing. *2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*. New Orleans, Lousiana, USA, 90–100.

PERKINS, D. D. et HUGHES, H. D. (2002). A survey on quality-of-service support for mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 2, 503–513.

RAJAGOPALAN, S. et SHEN, C.-C. (2006). A cross-layer decentralized bittorrent for mobile ad hoc networks. *MOBIQUITOUS 2006*. San Jose, Californie, USA.

S. CORSON, J. M. (1999). *RFC2501 : Mobile Ad hoc Networking (MANET) : Routing Protocol Performance Issues and Evaluation Considerations*. IETF.